Padrões de Projeto no dia a dia: Exemplos de uso

Clube Delphi

Fast Report na prática
Um tutorial de seus recursos para elaboração de relatórios

Ano 13 :: Edição 145 :: R$ 11,90

DevMedia

MULTICAMADAS

Saindo do Cliente/Servidor

InterBase XE
Fique por dentro do banco de dados da Embarcadero

Aplicando Segurança
Três implementações para controle de usuários

ISSN 1517990-7

Conhecimento faz diferença!

+ de 200 vídeos para assinantes

Faça já sua assinatura digital ! | www.devmedia.com.br/es

Faça um *upgrade* em sua carreira

Em um mercado cada vez mais focado em qualidade, ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre ouutros, pode ser a diferença entre conquistar ou não uma boa posição profissional.
Sabendo disso a DevMedia lançou uma publicação totalmente especializada em Engenharia de Software.
Todos os meses você pode encontrar artigos sobre Metodologias Ágeis; Metodologias tradicionais (document driven); ALM (application lifecycle); SOA (aplicações orientadas a serviçoes); Análise de sistemas; Modelagem; Métricas; Orientação à Objetos; UML; testes e muito mais. **Assine Já!**

DevMedia

Sumário

Delphi

Delphi

06 – Migrando de Cliente-Servidor para Multicamadas
[*Daniel Sobrinho Laporte*]

Banco de Dados, Minicurso

16 – InterBase XE – Parte 2
[*Fabrício Hissao Kawata*]

Delphi - Easy

24 – Aplicando Segurança
[*Giuliano Scombatti Pinto*]

Minicurso

36 – Fast Report na prática – Parte 1
[*Giuliano Scombatti Pinto*]

Boas Práticas

Boas Práticas

44 – Padrões de Projeto no dia a dia
[*Paulo Quicoli*]

Olá, eu sou o **DevMan**! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:
www.devmedia.com.br/clubedelphi/feedback

RENOVE JÁ!

Sua assinatura pode estar acabando

Renovando a assinatura de sua revista favorita você ganha brindes e descontos exclusivos.

www.devmedia.com.br/renovacao ou (21)3382-5038

DevMedia

Clube Delphi

Ano 13 - 145ª Edição 2012 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral

Paulo Quicoli
pauloquicoli@gmail.com

Editor Técnico

Daniel Sobrinho Laporte
daniel.laporte@gmail.com

Equipe Editorial

Guinther Pauli
guintherpauli@gmail.com

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br

Giuliano Scombatti Pinto
giuliano@sygnux.com.br

Jornalista Responsável

Kaline Dolabella - JP24185

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

www.devmedia.com.br/central
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

publicidade@devmedia.com.br

Distribuição

FC Comercial e Distribuidora S.A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

EDITORIAL

Nesta edição temos assuntos para o desenvolvedor Delphi que pensa em melhorar seus aplicativos. É muito comum encontrar desenvolvedores Delphi que possuem sistemas rodando há mais de cinco anos, utilizando arquitetura cliente/servidor. Há algum problema nisso? Claro que não, contudo, o mercado hoje está dinâmico, requer informações disponíveis em qualquer lugar e a qualquer momento. Como atualizar a estrutura de um aplicativo cliente/servidor para que disponibilize na web uma funcionalidade existente, sem duplicar código? A resposta é migrar para o modelo multicamadas. O Daniel apresenta um artigo que irá explicar esse processo de migração na prática, discutindo boas práticas. E como o assunto é alterar algo que já existe, porque não trocar seu gerador de relatório pelo Fast Report, que agora é parte integrante o Delphi XE 2?

O Giuliano inicia um minicurso onde serão abordados recursos que justificam a possibilidade dessa troca, vocês verão como é simples a criação de relatório multinível. O Fabrício encerra nesta edição sua abordagem dos recursos do Interbase XE, mostrando os componentes de acesso disponíveis e até mesmo como fazer uso de recursos administrativos disponíveis pelo SGBD. Além claro, de demonstrar como utilizar também via DbExpress.

O cuidado com a segurança das informações de um aplicativo é incontestável. Além de integridade física, existe também o nível de acesso a determinadas informações. Em minha opinião é praticamente inconcebível hoje um sistema comercial que não ofereça um controle de usuários. Assim, muitos acabam criando suas próprias rotinas de gerenciamento, talvez porque acreditem que não tenha nada pronto que possa satisfazer seus requisitos. O Giuliano mostra na prática como é trabalhoso fazer esse controle de forma manual, bloqueando acesso a controles visuais e tudo mais. Na sequência ele apresenta o componente User Control para XE 2, mostrando como seu uso agiliza a implementação desses controles e para finalizar seu artigo, é feita a introdução de um componente não muito conhecido pela comunidade, mas que está ganhando espaço, que é o TMS Security System. Ele funciona de uma forma um pouco diferente do User Control, principalmente em relação ao banco de dados que será utilizado para armazenar os dados da segurança. Não perca.

Para encerrar esta edição eu apresento um artigo sobre padrões de projeto. Mas desta vez com algo que vem de encontro com uma dúvida que talvez você leitor já tenha feito: "Onde eu posso aplicar um padrão de projeto?". Abordo os pilares dos projetos, pois eles não são apenas receitas a serem seguidas, existe um porquê de seu uso, tudo fundamentado em boas práticas e chego à demonstração de três situações comuns que podem ser melhoradas e que acontecem em qualquer sistema. Não importa se ele é orientado a objeto ou se é procedural, é mostrado que é possível aplicar boas práticas em pontos estratégicos de qualquer aplicação.

Dessa forma, boa leitura!

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!

Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Paulo Quicoli - Editor da Revista
pauloquicoli@gmail.com

Paulo Quicoli
pauloquicoli@gmail.com
Editor Geral
quicoli.wordpress.com
twitter.com/pauloquicoli

MVP

Assine agora e tenha acesso a todo o conteúdo da DevMedia:
www.devmedia.com.br/mvp

DevMedia

CURSOS ONLINE

Clube Delphi

A Revista Clube Delphi oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

CONHEÇA ALGUNS DOS CURSOS

- **Rad Studio 2007-DBX4 e Firebird 2.0**
  Aplicação comercial de vendas (básica) do Início ao Fim
- **Dominando a Orientação a Objetos e Componentes em Delphi (Avançado)**
- **Novidades** no Delphi 2010
- **Curso Loja Virtual em Delphi Prism**
- **Aplicação Financeira com Delphi 2009 e MySQL**

Para mais informações:

www.devmedia.com.br/cursos/delphi

(21) 3382-5038

http://www.devmedia.com.br

DevMedia

Migrando de Cliente-Servidor para Multicamadas

Permitindo o acesso a qualquer distância

Resumo DevMan

De que se trata o artigo:

O artigo trata de demonstrar de uma maneira clara a utilização das tecnologias Cliente-Servidor e Multicamadas, realizando comparativos e ilustrando as vantagens de se utilizar multicamadas no desenvolvimento de novas aplicações.

Em que situação o tema é útil:

O tema é útil em situações em que o leitor já possui sistemas desenvolvidos com a tecnologia Cliente-Servidor, porém, deseja migrar para o ambiente Multicamadas. Neste caso, o artigo trata os principais pontos de dúvida sobre a migração e suas principais vantagens.

Migrando de Cliente-Servidor para Multicamadas:

O assunto do momento é conectividade. Como tornar sua aplicação capaz de se comunicar com ambientes heterogêneos sem grandes esforços e com alta performance e disponibilidade? A resposta pode ser mais simples do que parece. Neste artigo iremos ver como proceder para que sua aplicação Delphi projetada na arquitetura Cliente-Servidor seja modificada a fim de atender as mais novas necessidades do mercado, utilizando o que há de mais novo no seu RAD Studio.

Até um bom tempo atrás, desenvolver aplicações apenas para ambiente Desktop era praticamente uma unanimidade e justamente nessa época a arquitetura Cliente-Servidor reinou, sendo adotada pelos desenvolvedores Delphi em sua grande maioria. Até então, todo o acesso ao banco de dados era realizado via rede local, com aplicações sendo executadas em ambientes Windows e certas vezes em Linux (seja por Kylix, Lazarus ou mesmo através de emulação), porém, os tempos mudaram. As aplicações que antes eram simplesmente acessadas em ambientes internos (dentro da própria empresa) vêm sendo refatoradas ou reescritas com a finalidade de atender as novas exigências de mercado.

No nosso novo quadro de requisitos, os clientes/usuários desejam acessar as aplicações via Web por meio de páginas, consultar e enviar informações através de Web Services, comunicar-se com aplicações desenvolvidas em outras plataformas, realizar integrações constantes de maneira transparente e rápida e mais recentemente, a utilizar Tablets e Smartphones. Cada vez mais a presença deles pode ser notada. Podemos citar seus usos frequentes em restaurantes ou deliveries onde os garçons realizam pedidos diretamente da mesa, sem a necessidade de se deslocar ao caixa. Este pedido já é automaticamente enviado para a cozinha, onde o prato é preparado e o tempo de espera é cada vez menor. Outra situação que ilustra este quando refere-se à parte financeira. Já é parte do cotidiano que os gestores consultarem informações do fluxo financeiro da empresa diretamente por seus Smartphones, independente do local, pois muitas vezes estão em viagem ou entre as mais variadas diversidades. A partir disto, surge principal pergunta abordada por este artigo: "Sua aplicação está preparada para suprir estas novas exigências?".

Desde os primórdios do Delphi fomos introduzidos aos conceitos de aplicações "online", podendo citar como exemplo algumas das tecnologias mais utilizadas na época (muitas delas em uso até hoje) como o IntraWeb (para criação de páginas), DCOM, COM+ e BSS (para aplicações distribuídas, ver **Nota do DevMan 1**) e ultimamente, mas não tão recente assim, as apresentações do Prism, RAD PHP e o reformulado e novo DataSnap.

Mesmo com todas estas tecnologias, vários desenvolvedores continuam com suas aplicações escritas em arquiteturas cliente-servidor, ou em outros casos, criando novas aplicações do início através desta mesma arquitetura, adotando ainda Delphi 5 ou 7 e consequentemente deixando de aproveitar muitos dos benefícios das últimas versões.

A Embarcadero desde o lançamento do Delphi 2009 vem investindo cada vez mais no desenvolvimento de aplicações "online" através do DataSnap, como o tema da vez é conectividade, todas suas atenções desde o RAD XE2 estão voltadas à possibilidade de reutilizar seu conhecimento em Object Pascal para a criação de Softwares para as mais variadas plataformas. Podemos citar a introdução a criação de aplicativos para iPhone por meio do FireMonkey, Windows Phone por meio do Prism XE2 ou Android, através do RAD PHP como vimos recentemente em uma excelente série de artigos publicados pelo Giuliano, aqui mesmo na Clube Delphi. Isto sem considerar a facilidade de integração com outros ambientes através do JSON, também introduzido a partir da versão 2009 (ver **Nota do DevMan 2**).

Com o passar do tempo e a chegada das novas versões, novos conceitos são apresentados de forma cada vez mais simples, de maneira que hoje para criar a base de uma aplicação multicamadas do início, bastam alguns cliques e já é possível testar métodos, comunicação com as portas e etc. Deste modo, devido à tamanha simplicidade e todas as vantagens dadas (que serão abordadas a seguir), não há justificativa para iniciar aplicações em outra arquitetura que não seja multicamadas. Deste modo o presente artigo tem por finalidade servir como um guia para desenvolvedores que desejam migrar suas aplicações ou mesmo iniciar novos projetos adentrando neste modelo de desenvolvimento.

Comparativo entre as arquiteturas

A arquitetura C-S (cliente-servidor, como vamos nos referir daqui em diante) segue um modelo baseado em duas camadas. Neste caso a primeira camada é representada pelo banco de dados e a segunda refere-se aos aplicativos que se comunicam diretamente com a camada de dados. O maior problema deste modelo de arquitetura diz respeito à manutenção dos projetos. Muitas vezes, uma simples alteração que demanda apenas a adição de um novo campo interno ou mesmo mudança das regras de negócio faz com que seja necessária a compilação de todo o projeto, fazendo necessária a distribuição do Software para atualização de todos os usuários do sistema. Em muitos casos os desenvolvedores optam por realizar a modularização da aplicação por meio de BPLs, e, embora esta solução ajude em alguns casos, ainda assim será necessária a distribuição das partes alteradas para os possíveis clientes.

Diferentemente do modelo C-S, a arquitetura multicamadas promove uma maior independência entre as partes do projeto, desacoplando a camada responsável por dados das camadas visuais. Em situações como esta anteriormente mencionada, apenas a aplicação servidora necessitaria ser atualizada, fazendo com que isto fique "transparente" ao seu cliente. Esse sem dúvida nenhuma é um dos grandes benefícios/diferenciais da escolha deste modelo e que influencia diretamente na manutenção do projeto. Entre várias outras vantagens em relação a arquitetura convencional, podemos citar as principais que referem-se a distribuição geográfica (possibilidade de trabalhar de qualquer localidade com alta performance), separação das regras de negócio e interface (GUI – ver **nota do DevMan 3**), facilidade de manutenção como já mencionado, além da comunicação com outras tecnologias/ambientes de maneira simples e sem grandes esforços.

Para exemplificar uma situação mais próxima da realidade, desenvolveremos duas aplicações. Sendo assim, será criada uma aplicação de agenda primeiramente em duas camadas, em seguida, faremos sua conversão para o ambiente multicamadas, o que justifica de fato o desenvolvimento do exemplo escolhido já que torna-se possível acessá-la de qualquer local e por meio dos mais variados dispositivos.

Durante este processo iremos comentar situações peculiares de cada arquitetura e que requerem uma atenção maior por parte do desenvolvedor, tornando o exemplo escolhido o mais próximo de uma aplicação real. Também veremos várias dicas para uma boa compreensão do conceito.

Dadas as devidas comparações vamos dar início ao desenvolvimento do exemplo tomando como primeira etapa a escolha do SGDB. Esta é uma questão bastante pertinente ao desenvolvimento, e que sem dúvida é fonte de grandes discussões. Embora não seja uma regra para o desenvolvimento de aplicações distribuídas, minimizar a dependência do Software e SGDB pode trazer benefícios. Isso porque caso seja adotada uma solução de multiconectividade como o dbExpress, ZeosDBO entre outras, o sistema estará não só apto a funcionar de qualquer localidade

Nota do DevMan 1

DCOM (Distributed Component Object Model), COM (Component Object Model) e COM + são tecnologias arquitetadas pela Microsoft que viabilizam a comunicação entre aplicações client/server, realizada através de interfaces. O BSS (Borland Socket Server) é uma solução baseada em Sockets na qual utiliza a tecnologia COM+ para o desenvolvimento de aplicações multicamadas. Diferentemente do COM+, utiliza um executável como servidor de aplicação ao invés de uma DLL. Já o Intraweb é uma tecnologia que utiliza o desenvolvimento para Web baseada em CGIs.

Nota do DevMan 2

O JSON (Javascript Object Notation) é uma espécie de padronização para transporte de informações entre ambientes heterogêneos e que tem como linguagem o JavaScript.

Nota do DevMan 3

GUI (Graphical User Interface) refere-se às interfaces que promovem interatividade entre o usuário e os dispositivos na qual se baseiam em elementos gráficos como ícones entre outros indicadores visuais.

como também em qualquer banco de dados, trazendo ainda mais flexibilidade na gama de plataformas a serem atendidas pela mesma solução. De qualquer modo, como já mencionado, o objetivo é termos um ambiente próximo da realidade do leitor, sendo assim, iremos utilizar recursos do SGDB escolhido, justamente para poder comentar os pontos chave a serem revistos.

Tutorial

Criando o banco de dados de exemplo

Para iniciar o exemplo iremos criar um banco de dados de exemplo que simule um simples relacionamento "Pai-Filho" ou "um para muitos". Neste caso foi utilizado o SGDB Firebird 2.5 (disponível na sessão **Links**). Como comentado anteriormente, lembre-se de que se deseja uma aplicação flexível, uma boa escolha pode ser minimizar a dependência entre aplicação e banco de dados. O objetivo de optar por este caminho é simplesmente para efeito de tornar menos traumático o uso do seu projeto com outros bancos de dados. De qualquer modo, utilizaremos alguns recursos específicos do Firebird exatamente para tornar o nosso cenário o mais próximo do real, no entanto, estes recursos serão comentados, expondo outras maneiras de se realizar a mesma tarefa (veremos isto mais adiante).

Como se pode notar na **Figura 1**, a modelagem da nossa aplicação é muito simples e conta com duas tabelas (Contato e Telefone). A **Listagem 1** contém o script SQL para a elaboração das tabelas do projeto.

Listagem 1. Script das tabelas

```
CREATE TABLE CONTATO (
  IDCONTATO   INTEGER NOT NULL,
  NOME        VARCHAR(250) NOT NULL,
  ENDERECO    VARCHAR(250) NOT NULL,
  BAIRRO      VARCHAR(250) NOT NULL,
  CIDADE      VARCHAR(250) NOT NULL,
  ESTADO      CHAR(2) NOT NULL,
  NUMERO      INTEGER NOT NULL
);

ALTER TABLE CONTATO ADD PRIMARY KEY (IDCONTATO);

CREATE TABLE TELEFONE (
  IDTELEFONE      INTEGER NOT NULL,
  IDCONTATO       INTEGER NOT NULL,
  TELEFONE        VARCHAR(15) NOT NULL,
  TIPO_TELEFONE   CHAR(1) NOT NULL
);

ALTER TABLE TELEFONE ADD PRIMARY KEY (IDTELEFONE);

ALTER TABLE TELEFONE
ADD CONSTRAINT FK_TELEFONE_CONTATO
FOREIGN KEY (IDCONTATO) REFERENCES CONTATO (IDCONTATO)
ON DELETE CASCADE
ON UPDATE CASCADE
USING INDEX IDX_TELEFONE_CONTATO;

CREATE SEQUENCE GEN_IDTELEFONE;
CREATE SEQUENCE GEN_IDCONTATO;
```

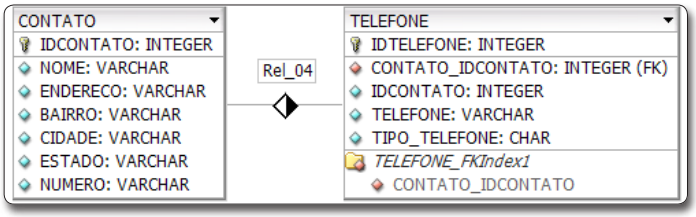

Figura 1. Modelagem do banco de dados de exemplo

Criado o banco de dados, passaremos ao desenvolvimento da aplicação na arquitetura Cliente-Servidor.

Criando a aplicação Cliente-Servidor

Para iniciarmos o projeto, crie uma nova aplicação VCL Forms (File>New>VCL Forms Application – Delphi). Ao formulário inicial, atribua o nome de frmPrincipal e salve em seguida, nomeando a unit como untPrincipal.pas e o projeto como Agenda.dproj. Adicione a seguir um novo formulário (File>New>VCL Form – Delphi) nomeado como frmPesquisa (untPesquisa.pas). Por último, insira um novo Data Module (File>New>Other>Delphi Files>Data Module), este será o DMPrincipal (untDMPrincipal.pas). Com isso, temos a base para a criação da aplicação de agenda utilizando a arquitetura C-S.

Iniciaremos por algo que em uma aplicação C-S é utilizado para a organização dos dados, o Data Module. Embora uma de suas funções seja a organização e centralização do que refere-se à acesso a dados, muitas vezes, os desenvolvedores optam por criar um ou dois Data Modules, concentrando as regras de dados nos próprios formulários. Esta é uma situação bastante comum no cotidiano do desenvolvedor Delphi, porém, esta prática promove não só a desorganização como a dificuldade de manutenção, pois parte das regras encontra-se no formulário visual e parte na camada de dados, o que quase sempre ocasiona "bad smells" nos projetos (ver **Nota do DevMan 4**).

Nota do DevMan 4

Bad smell ("mau cheiro") é um termo utilizado na refatoração de Software na qual se refere ao código que apresenta problemas (deve ser refatorado). Segundo a refatoração, esse tipo de código é geralmente caracterizado por situações como duplicidade ou métodos muito longos, por exemplo, o que ocasiona dificuldades no processo de manutenção e que certamente deve ser observado com a finalidade de ser otimizado.

Sendo assim, configuraremos os componentes referentes ao acesso a dados. Adicione ao DMPrincipal 1 SQLConnection, 2 SQLDataSet, 2 DataSetProvider, 1 DataSource e 3 ClientDataSet. Configure-os conforme exibido na **Figura 2**. Lembrando que mais adiante no momento da criação da aplicação multicamadas, iremos modificar esse Data Module atual, separando parte dos seus componentes e regras em uma camada extra. Por enquanto, apenas atente as configurações em questão.

Após adicionar os componentes mencionados, é hora de realizar a comunicação com a outra camada desta arquitetura (banco de

dados). Para isto, crie uma nova conexão com o Firebird utilizando o Data Explorer (View>Data Explorer). Feito isso, configure o SQLConnection apontando sua propriedade ConnectionName para a configuração previamente criada, não esquecendo de desmarcar a opção LoginPrompt para que o login não seja requisitado a cada conexão. Nos componentes SQLDataSet aponte as propriedades SQLConnection para o SQLConexao e adicione as instruções da **Listagem 2** aos respectivos CommandText.

Listagem 2. Instruções SQL dos componentes SQLDataSet

```
[sqlAgenda]
SELECT
  CONTATO.IDCONTATO,
  CONTATO.NOME,
  CONTATO.ENDERECO,
  CONTATO.BAIRRO,
  CONTATO.CIDADE,
  CONTATO.ESTADO,
  CONTATO.NUMERO
FROM
  CONTATO
WHERE
  CONTATO.IDCONTATO =:IDCONTATO

[sqlTelefones]
SELECT
  TELEFONE.IDTELEFONE,
  TELEFONE.IDCONTATO,
  TELEFONE.TELEFONE,
  TELEFONE.TIPO_TELEFONE
FROM
  TELEFONE
WHERE
  TELEFONE.IDCONTATO =:IDCONTATO

[sqlPesquisa]
SELECT
  CONTATO.IDCONTATO,
  CONTATO.NOME
FROM
  CONTATO
```

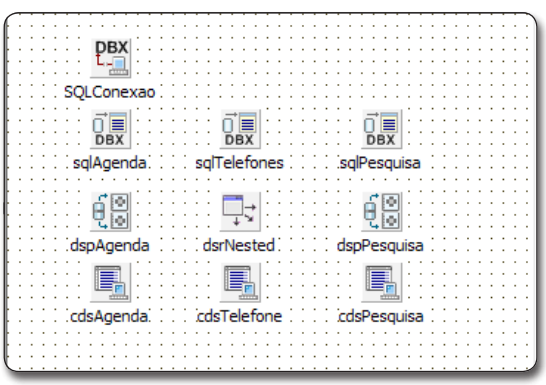

Figura 2. Componentes do DMPrincipal

Nota

Para não estender muito o artigo, as demais configurações dos DataSetProviders e ClientDataSets não serão abordadas, porém, a aplicação completa está disponível para download no portal da DevMedia.

Configurados os componentes, passaremos para a parte de codificação. O primeiro método a ser criado é a função getSQLOriginal que deve ser criada no escopo public do Data Module e tem seu código detalhado a seguir na **Listagem 3**.

O objetivo da função getSQLOriginal como o próprio nome sugere é retornar o SQL original para a consulta dos contatos da agenda. Esta instrução será utilizada no momento da criação do formulário de pesquisa, realizando a concatenação das condições de forma dinâmica, conforme será mostrado mais adiante.

O próximo passo é criar a rotina responsável por obter um valor de chave primária válido aos respectivos campos chave. Para isso é definida a função getValorChave desta vez no escopo private do DM. Seu código pode ser visto na **Listagem 4.**

Listagem 3. Função getSQLOriginal

```
01 function TDMPrincipal.getSQLOriginal: String;
02 begin
03   Result := ' SELECT CONTATO.IDCONTATO, CONTATO.NOME FROM CONTATO ';
04 end;
```

Listagem 4. Função getValorChave do DMPrincipal

```
01 function TDMPrincipal.getValorChave(NomeCampo: String): Integer;
02 var
03   sql : TSQLDataSet;
04 begin
05   sql := TSQLDataSet.Create(Self);
06   try
07     sql.SQLConnection := SQLConexao;
08     sql.CommandText :=
09       'SELECT GEN_ID(GEN_' + NomeCampo + ',1) FROM RDB$DATABASE';
10     sql.Open;
11     Result := sql.Fields[0].Value;
12   finally
13     sql.Close;
14     FreeAndNil(sql);
15   end;
16 end;
```

O objetivo é obter um valor para as chaves primárias. Deste modo é utilizado um SQLDataSet criado em Runtime para realizar esta requisição (linha 05). Note que após a atribuição da conexão (linha 06), é montada uma instrução SQL utilizando a função GEN_ID (linhas 08 e 09). Esta função, conforme citado no tópico de criação do banco de dados, é um recurso específico do Firebird e, portanto, faz com que a aplicação esteja dependente do SGDB. Através do parâmetro de entrada "NomeCampo" (autoexplicativo), é chamado um objeto Generator previamente criado no SGDB (ver script da **Listagem 1**). Esta é uma situação que poderia ser substituída por algum tipo de solução que não fizesse uso de recursos específicos do banco. Entre algumas alternativas para este caso, poderíamos citar a criação de uma classe que por meio de uma propriedade, identificasse o banco de dados que está sendo utilizado no momento, criando deste modo, vários overloads da função mencionada e consequentemente tomando ações diferentes em cada ocasião. Outra solução para isso seria criar tabelas que

armazenassem os valores dos respectivos campos chaves, fazendo um incremento "manual" na própria aplicação, dando assim mais autonomia à sua solução. Continuando a explanação do código da **Listagem 4**, na linha 11 o valor do Generator é obtido diretamente do banco e retornado, fechando a conexão e liberando o componente da memória (linhas 13 e 14). Com isso está pronta a função getValorChave.

Finalizando o DMPrincipal, alimentaremos os valores iniciais de cada DataSet (cdsAgenda e cdsTelefone) através dos eventos OnNewRecord e utilizando para isso a função anteriormente definida. Observe o código da **Listagem 5**.

Listagem 5. Eventos OnNewRecord dos ClientDataSets

```
01 procedure TDMPrincipal.cdsAgendaNewRecord(DataSet: TDataSet);
02 begin
03   cdsAgendaIDCONTATO.Value := getValorChave('IDCONTATO');
04 end;

05 procedure TDMPrincipal.cdsTelefoneNewRecord(DataSet: TDataSet);
06 begin
07   cdsTelefoneIDTELEFONE.Value := getValorChave('IDTELEFONE');
08   cdsTelefoneTIPO_TELEFONE.AsString := 'R';
09 end;
```

Note na utilização da função getValorChave que o parâmetro é o nome do próprio campo chave. Com a parte de acesso a dados definida e funcionando, passaremos para a criação das telas, priorizando o formulário de pesquisa.

Abra o frmPesquisa e adicione um componente Panel, Label, ComboBox, Edit, BitBtn, DBGrid e DataSource. Ao final do processo, configure o Layout conforme o demonstrado na **Figura 3.** Renomeie o BitBtn para btnPesquisar, ComboBox para cmbParametro e o Edit para edtPesquisa. Adicione à propriedade Items do ComboBox os valores que serão utilizados para pesquisa (Código e Nome). Apenas configure o DBGrid ao DataSource através da propriedade de mesmo nome e adicione as respectivas colunas. Adicione ao uses do formulário a unit untDMPrincipal e por último, aponte o dsrPesquisa para o componente sqlPesquisa do Data Module. Feito isso as configurações do formulário estarão prontas e passaremos para a codificação. Vale lembrar novamente que o objetivo é transformar a aplicação cliente-servidora em uma aplicação multicamadas, portanto, futuramente este formulário sofrerá mudanças. A dica é sempre observar nos pontos relevantes de cada arquitetura, embora estes serão comentados a seguir.

A primeira implementação realizada será no evento OnDblClick da Grid (**Listagem 6**). Com isto faremos que ao realizar um duplo clique na DBGrid, o formulário seja fechado com o ModalResult = mrOk. Isto para que caso seja selecionado um registro na pesquisa, o cadastro seja automaticamente aberto na tela principal da agenda.

O próximo passo é fazer com que a cada vez que for realizada uma pesquisa, os ClientDataSet de pesquisa não fique aberto desnecessariamente. Embora este trabalhe de forma desconectada do banco de dados, evitando conexões abertas, ainda assim utilizará memória para armazenar os dados. Sendo assim, utilizamos o evento OnDestroy do formulário para fechá-lo, conforme a **Listagem 7.**

A implementação final da tela de pesquisa refere-se ao evento OnClick do botão pesquisar. O código é ilustrado na **Listagem 8.**

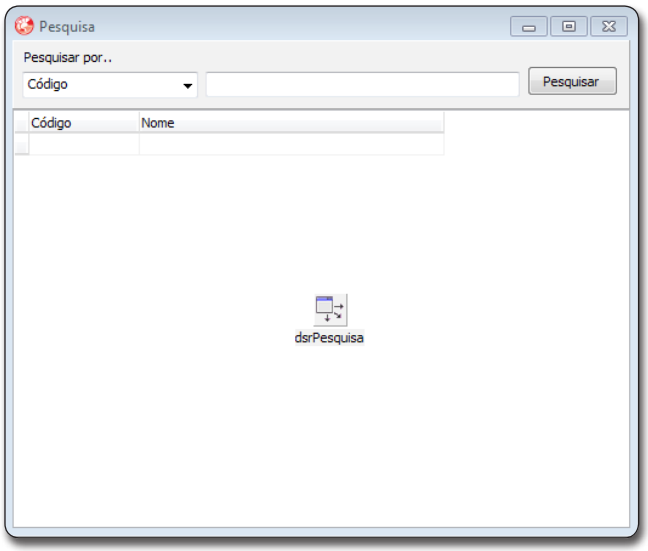

Figura 3. Layout do formulário de pesquisa

Listagem 6. Evento OnDblClick da DBGrid

```
01 procedure TfrmPesquisa.DBGrid1DblClick(Sender: TObject);
02 begin
03   Self.ModalResult := mrOk;
04 end;
```

Listagem 7. Evento OnDestroy do frmPesquisa

```
O1 procedure TfrmPesquisa.FormDestroy(Sender: TObject);
02 begin
03   DMPrincipal.cdsPesquisa.Close;
04 end;
```

Listagem 8. Evento OnClick do botão de pesquisa

```
01 procedure TfrmPesquisa.btnPesquisaClick(Sender: TObject);
02 begin
03   DMPrincipal.sqlPesquisa.Close;
04   if Trim(edtPesquisa.Text) = '' then
05     DMPrincipal.sqlPesquisa.CommandText := DMPrincipal.getSQLOriginal
06   else
07   begin
08     if cmbParametro.ItemIndex = 0 then
09       DMPrincipal.sqlPesquisa.CommandText := DMPrincipal.getSQLOriginal +
       ' WHERE IDCONTATO = ' + Trim(edtPesquisa.Text)
10     else
11     DMPrincipal.sqlPesquisa.CommandText := DMPrincipal.getSQLOriginal +
       ' WHERE NOME LIKE ' + QuotedStr(Trim(edtPesquisa.Text)) + ' || ' +
       QuotedStr('%');
12   end;
13   DMPrincipal.cdsPesquisa.Close;
14   DMPrincipal.cdsPesquisa.Open;
15 end;
```

Na linha 03 o componente sqlPesquisa é fechado para receber a instrução SQL com as devidas condições. Caso não tenha sido digitado nenhum conteúdo no Edit, a pesquisa é realizada considerando tudo (linhas 04 e 05). Caso contrário, é verificado se foi selecionado o conteúdo referente ao código do contato (linhas 08 e 09). Na situação onde é escolhido o nome do contato, a instrução é montada utilizando o operador LIKE (linhas 11 e 12). Note que em todos os casos foi utilizada a função getSQLOriginal para a obtenção da instrução original de SQL. Finalmente nas linhas 13 e 14 o ClientDataSet é aberto, retornando os dados. Aqui vale lembrar um problema muito comum nas aplicações cliente-servidor, a presença de código SQL no formulário. Esta não é uma boa prática, pois mistura código pertencente à camada de dados com a GUI. Neste tipo de situação pode-se optar pela aplicação de um Pattern como o MVC, por exemplo, (ver **Nota do DevMan 5**), na qual traz como grandes benefícios a maior organização do código, facilidade de manutenção e possibilidade de elaboração de testes automatizados. Embora a aplicação de Design Patterns seja altamente recomendada, muitos desenvolvedores Delphi deixam de utilizá-los e adotam o desenvolvimento procedural, alegando a perda do RAD.

Nota do DevMan 5

MVC é a sigla de Model-View-Controller e refere-se a um dos mais conhecidos e utilizados Patterns. Diferente da ideia de camadas abordada no conceito de multicamadas que se refere à aplicações geograficamente distribuídas, o MVC prega a independência entre as camadas de uma mesma aplicação, onde temos Model como a representação dos dados ou das classes de Domínio, View (visão) referenciando as interfaces GUI e por último o Controller, que é o "intermediador" da comunicação entre suas camadas.

Para a finalização da aplicação C-S, faremos a tela principal da aplicação de agenda. O objetivo é tornar esta mesma tela funcional para as duas arquiteturas, utilizando assim o mesmo código fonte. Para isso, trabalharemos diretamente com o DataSource quer será inserido para cadastro (dsrAgenda). Os eventos dos botões contendo as ações (incluir, excluir, cancelar, salvar e etc) serão todas implementadas utilizando diretamente o DataSource. Desta forma, os eventos irão ocorrer mesmo que a fonte de dados seja alterada (outro ClientDataSet diferente do cdsAgenda). O primeiro passo é adicionar os componentes conforme o layout da **Figura 4**.

Para não estender muito o artigo, os códigos referentes aos botões (ações do ActionList) não serão vistos por completo, abordando apenas as principais partes do exemplo, porém, pode-se encontrar o código completo do exemplo para download no site da DevMedia.

Considerando que os componentes já estão devidamente configurados, faremos a primeira implementação que se refere ao evento OnCreate do formulário na qual realizada a abertura do DataSet conectado ao DataSource, conforme mostra a **Listagem 9**.

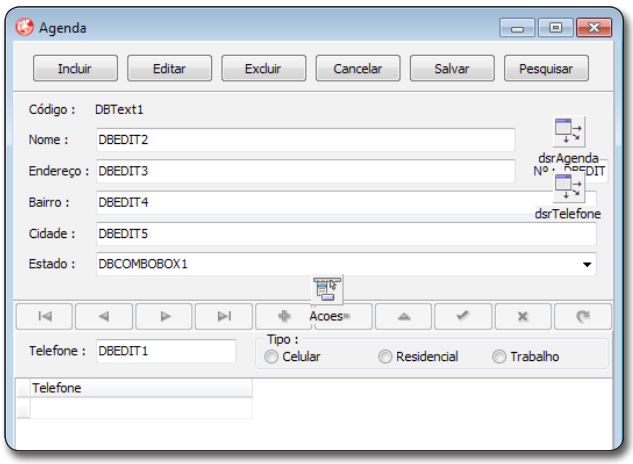

Figura 4. Layout da aplicação de agenda

O evento OnUpdate do ActionList realizará o controle das ações do formulário. Deste modo, a cada mudança de estado do DataSource os botões ficarão habilitados ou não. O código é exibido na **Listagem 10**.

Para finalizar os eventos do formulário principal iremos abordar a ação actPesquisar, que como o próprio nome sugere, faz referência ao código utilizado para a consulta dos contatos. O código completo encontra-se na **Listagem 11**.

Listagem 9. Evento OnCreate do form

```
01 procedure TfrmPrincipal.FormCreate(Sender: TObject);
02 begin
03  if not dsrAgenda.DataSet.Active then
04   dsrAgenda.DataSet.Open
05 end;
```

Listagem 10. Evento OnUpdate do ActionList

```
01 procedure TfrmPrincipal.AcoesUpdate(Action: TBasicAction;
    var Handled: Boolean);
02 begin
03  actIncluir.Enabled  := not (dsrAgenda.State in [dsEdit,dsInsert]);
04  actEditar.Enabled   := not (dsrAgenda.State in [dsEdit,dsInsert]);
05  actCancelar.Enabled := dsrAgenda.State in [dsEdit, dsInsert];
06  actSalvar.Enabled   := dsrAgenda.State in [dsEdit, dsInsert];
07  actExcluir.Enabled  := dsrAgenda.State in [dsEdit, dsInsert];
08 end;
```

Listagem 11. Código da ação actPesquisar

```
01 procedure TfrmPrincipal.actPesquisarExecute(Sender: TObject);
02 begin
03  frmPesquisa := TfrmPesquisa.Create(Self);
04  with frmPesquisa do
05  try
06   if ShowModal = mrOk then
07   begin
08    dsrAgenda.DataSet.Close;
09    TClientDataSet(dsrAgenda.DataSet).FetchParams;
10    TClientDataSet(dsrAgenda.DataSet).Params[0].AsInteger :=
      dsrPesquisa.DataSet.Fields[0].AsInteger;
11    dsrAgenda.DataSet.Open;
12   end;
13  finally
14   FreeAndNil(frmPesquisa);
15  end;
16 end;
```

Podemos notar a chamada do formulário de pesquisa (linha 03). Na linha 06 o resultado da consulta é retornado através do ModalResult da tela de consulta. Caso seja mrOk, o DataSource é fechado (linha 08), os parâmetros são carregados e atribuídos, para que finalmente o cadastro seja aberto (linhas 09 a 11). Finalmente o formulário de pesquisa é destruído e liberado da memória (linha 14).

Criando a aplicação Multicamadas

Finalizada a aplicação cliente-servidor, passaremos à arquitetura nova. A primeira etapa se dará na elaboração do servidor de aplicação DataSnap. Para isso, inicie um novo projeto através do menu File>New>Other>DataSnap Server. Isso fará com que seja aberto o Wizard de criação de servidores de aplicação DataSnap. Na primeira tela é pedido para que se informe o tipo de servidor que se deseja criar VCL Application, Console ou um servidor baseado em serviços. Selecione a primeira opção e clique em Next. Na próxima tela são solicitadas as configurações que possibilitam comportamentos diferenciados à aplicação, tais como a disponibilização dos métodos do servidor por meio de REST. Caso queira habilitar essa funcionalidade, deve-se selecionar demais protocolos (HTTP - convencional e/ou HTTPS - com autenticação). A **Figura 5** ilustra a configuração utilizada no exemplo, após isto clique novamente em Next. A tela seguinte tem por finalidade apenas o teste de portas (conforme os protocolos que você selecionou). Neste caso, deixaremos a porta padrão 211. Se desejar, clique em "Test Port" para verificar se há outra aplicação utilizando a mesma, caso contrário prossiga clicando em Next. Por último, a tela final solicita qual será a classe utilizada como container da aplicação. Neste caso foi selecionada a classe base como TDSServerModule. Aqui vale lembrar que esta é uma convenção do modelo multicamadas que sugere uma camada Container para a centralização das regras e classes do servidor. Dadao o término do Wizard, finalmente clique em Finish e você terá um servidor DataSnap pré-configurado. Salve seu projeto como Servidor.dproj, o formulário como frmServidor (untServidor.pas) e o Container como srvContainer (untSrvContainer.pas). Execute a aplicação e veja que já é possível rodá-la na porta anteriormente configurada.

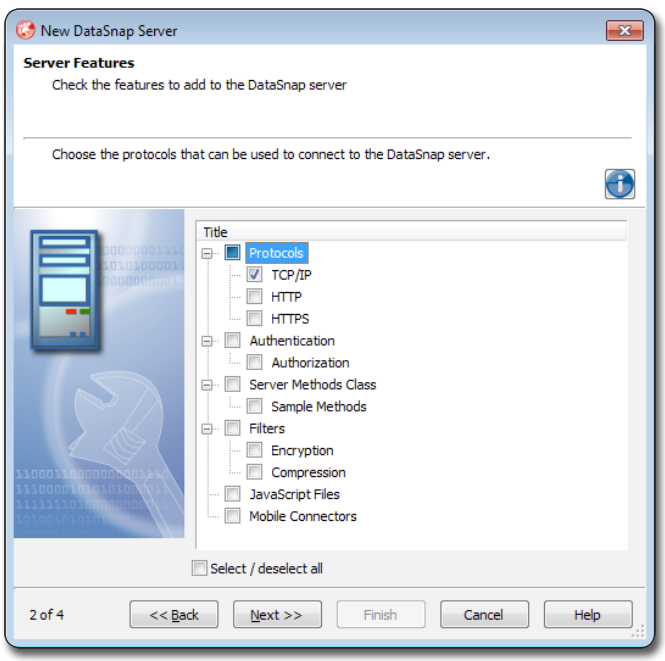

Figura 5. Configuração utilizada na construção do servidor de aplicação DataSnap

Como já mencionado e colocado em prática até aqui, a criação de servidores tornou-se uma tarefa simples e muito prática, bastando o mínimo de conhecimento sobre as tecnologias envolvidas, porém, nossa aplicação ainda não está pronta. O próximo passo é realizar a separação do que se refere à camada de acesso a dados no servidor de aplicação.

A primeira alteração a ser feita é criar a conexão com o banco. Desta forma, o componente SQLConexao que antes ficava no DM-Principal da aplicação C-S deve ser colocado junto ao Container e configurado para realizar a conexão com o BD da mesma forma de antes. Adicione também ao container um componente DSServerClass (SCPrincipal) conectando-o ao componente DSServer através

da propriedade de mesmo nome. Ressalto que este componente é o principal responsável por expor os métodos e conteúdos da camada servidora tanto para a aplicação cliente, quanto qualquer outra forma, como o próprio acesso via HTTP por exemplo. Ao final do processo seu container estará semelhante ao da **Figura 6**.

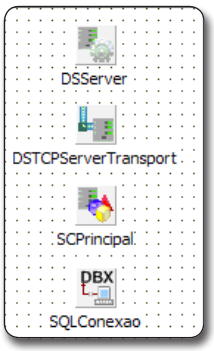

Figura 6. Container da aplicação servidora

Diferentemente da arquitetura C-S, aqui neste modelo temos a centralização de tudo em Server Modules. É através deles que a aplicação cliente "enxergará" os componentes e métodos do servidor realizando a comunicação direta de forma quase "transparente". Sendo assim, desvincularemos parte do que ficava "misturado" na aplicação C-S e que será representado pela camada cliente, passando para o Server Module. Para isso, crie um novo Server Module (File>New>Other>DataSnap Server>Server Module) e salve-o com o nome de SMPrincipal (untSMPrincipal.pas). Ao final deste processo daremos início ao processo de separação.

A primeira parte a ser transportada para o servidor de aplicação são os componentes. Como o próprio nome sugere, o Client DataSet é o único componente que deverá se manter na aplicação cliente. Deste modo, os responsáveis pela persistência (SQL DataSet e DataSetProvider) que fazem menção ao acesso a dados, deverão estar na aplicação servidora. Deste modo, proceda da mesma forma com que foram configurados os componentes na aplicação C-S, configurando-os para o SMPrincipal. Em seguida adicione a unit untSrvContainer ao uses do Server Module, configurando o SQLConexao para os SQLDataSets. O resultado é mostrado na **Figura 7.**

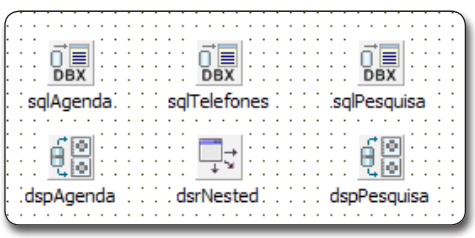

Figura 7. Server Module configurado

Quanto a parte de código, as funções getSQLOriginal e getValorChave são definidas aqui na sessão private e public respectivamente. Isto porque apenas este Server Module possuirá acesso as suas próprias consultas. O método getSQLOriginal não necessita de mudanças, portanto, pode ser considerado o código da **Listagem 3.** Quanto ao método getValorChave, pode-se manter o código da **Listagem 4** modificando apenas a linha 07 (referente a conexão), que deverá considerar o componente SQLConexao do container, conforme a seguir:

```
sql.SQLConnection := srvContainer.SQLConexao;
```

Repare que até aqui apenas deslocamos os componentes de lugar, porém, separando as camadas. O principal processo que difere entre as arquiteturas é o modo de pesquisa. No próprio exemplo, utilizamos a inserção de código SQL diretamente no formulário de consulta, o que de fato não é uma boa prática. Quanto a isso, em multicamadas temos toda a definição de consultas centralizada no servidor de aplicação. Isto porque caso seja necessário adicionar novos campos ou expor alguma nova regra, apenas o servidor sofrerá alteração (quando não houverem mudanças que reflitam na interface). Para essa situação pode-se usar duas possíveis soluções: a primeira é escrever métodos que sejam chamados na aplicação cliente através da Proxy (veremos adiante), já a segunda é utilizar os eventos do próprio DataSetProvider. A segunda opção foi adotada para nosso exemplo. Aqui utilizamos o evento OnDataRequest do dspPesquisa para elabora a consulta. Veremos mais adiante como se comunicar com este evento. O código encontra-se na **Listagem 12**.

Listagem 12. Evento OnDataRequest do dspPesquisa

```
// Adicione Variants ao uses
01 function TSMPrincipal.dspPesquisaDataRequest(Sender: TObject;
   Input: OleVariant): OleVariant;
02 begin
03  sqlPesquisa.Close;
04  if Input[0] = 0 then
05  begin
06   if Trim(VarToStr(Input[1])) = '' then
07    sqlPesquisa.CommandText := getSQLOriginal
08   else
09   sqlPesquisa.CommandText := getSQLOriginal +
     ' WHERE IDCONTATO = ' + VarToStr(Input[1]);
10  end
11  else if Input[0] = 1 then
12  begin
13 sqlPesquisa.CommandText := getSQLOriginal +
    ' WHERE NOME LIKE ' + QuotedStr(VarToStr(Input[1])) + ' || ' +
    QuotedStr('%');
14  end;
15  Result := dspPesquisa.Data;
16 end;
```

No código da **Listagem 12** podemos observar alguns detalhes interessantes. O primeiro é que o método é uma função que retorna um OleVariant (arrayList de Variants). O segundo é o parâmetro de entrada Input que faz referência também a um tipo OleVariant. Observe na linha 03 que o componente de pesquisa ainda é o sqlPesquisa. Por se tratar de um Array de Variants,

o parâmetro Input permite receber os mais variados valores em forma de ArrayList. Neste caso é checado o parâmetro da posição 0 que definirá o campo recebido por parâmetro (linha 04). Caso o Input[0] (primeira posição do array) seja 0, a consulta será realizada via código. Note que Input[1] (segunda posição do array) refere-se ao valor passado como parâmetro. Sendo o valor nulo (linha 06), a consulta será realizada de forma a retornar todos os registros, caso contrário, é optada uma consulta por código, concatenando a SQL com o valor do parâmetro Input[1] que conterá o IDContato vindo da aplicação cliente (linhas 07 a 10). Caso a opção seja por nome do contato, a instrução é montada com base no nome, utilizando também o valor de Input[1] (linhas 11 a 14). Desta forma poderíamos definir quantos "Inputs" desejássemos. Por último na linha 15, o Result da própria função recebe o valor de dspPesquisa.Data, na qual se encarregará de utilizar o sqlPesquisa como consulta, retornando o resultado através de seu método ".Data" que também é do tipo OleVariant. A grande vantagem é que aqui temos a definição do código SQL apenas no lado do servidor, bastando apenas passar parâmetros, já a desvantagem é que caso fossemos disponibilizar esta consulta através do servidor de aplicação e REST, necessitaríamos de criar um método específico para esta finalidade.

Com isso finalizamos a implementação do Server Module. Feito isso, basta disponibilizá-lo a camada cliente. Para isso utilizamos o evento OnGetClass do componente DSServerClass que se encarrega do trabalho para nós (**Listagem 13**).

Listagem 13. Evento OnGetClass do DSServerClass

```
//Adicione a untSMPrincipal ao uses
01 procedure TsrvContainer.SCPrincipalGetClass(DSServerClass:
   TDSServerClass; var PersistentClass: TPersistentClass);
02 begin
03   PersistentClass := TSMPrincipal;
04 end;
```

A **Listagem 13** demonstra o parâmetro de entrada Persistent Class que recebe a classe TSMPrincipal como conteúdo (linha 03), disponibilizando este Server Module para as demais camadas que necessitarem consumi-lo. Feito isso o servidor de aplicação está finalizado. Nos atentaremos agora as particularidades da aplicação Cliente.

Conforme dito, a camada cliente armazena os conteúdos responsáveis pela conectividade entre o servidor, além da interface. Nela podemos utilizar assim como em C-S os Data Modules para centralização e organização destes componentes. A partir disto, utilizaremos o DMPrincipal para esta finalidade.

Como nesta camada não há conectividade direta com o banco de dados, a nossa comunicação se dará logicamente com o servidor. Neste caso, o responsável por esta comunicação é o próprio SQLConnection, porém, configurado com o driver DataSnap operando na porta anteriormente definida (**Figura 8**).

Em relação aos componentes ClientDataSet que restaram, devemos fazer com que se comuniquem com suas respectivas fontes de dados (DataSetProviders), porém, para que isto ocorra, necessitamos de um comunicador, descendente da classe TCustomRemoteServer, para esta finalidade utilizamos o DSProviderConnection, apontando sua propriedade SQLConnection para o SQLConexao e informando a classe que foi definida no evento OnGetClass (TSMPrincipal no caso). Por último, basta conectar o cdsAgenda ao componente DSProviderConnection através da propriedade RemoteServer e os DataSetProviders estarão disponíveis. O mesmo deve ser feito para dspPesquisa e por último, cdsTelefone, que deverá ser conectado ao DataSetField de cdsAgenda, através da propriedade de mesmo nome. O resultado final do DMPrincipal é exibido na **Figura 9.**

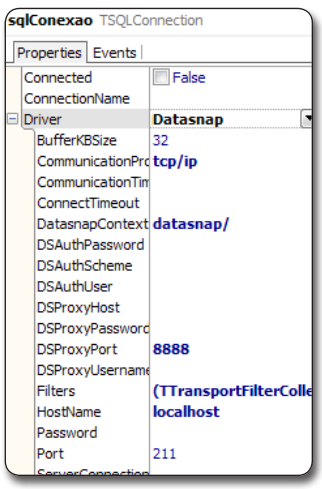

Figura 8. Configuração do SQLConexao para a conexão com o servidor

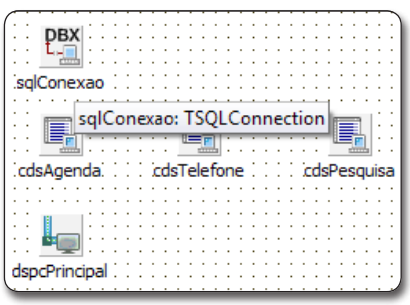

Figura 9. DMPrincipal na aplicação cliente

Para finalizar a aplicação cliente, faremos as devidas configurações restantes. A primeira refere-se à nova chamada dos campos chave primária através do método getValorChave. A diferença principal é que desta vez estamos fazendo uma chamada diretamente do servidor de aplicação. Para isto é necessário criar a unit ProxyClient. Sendo assim, execute o servidor de aplicação, clique de direita sobre o sqlConexao da camada cliente e selecione a opção "Generate DataSnap Client Classes". Este processo define uma classe que contém os métodos do servidor de aplicação e todas suas definições. Sendo assim, crie suas respectivas chamadas nos eventos OnNewRecord do cdsAgenda e cdsTelefone. O exemplo de utilização do cdsAgenda é ilustrado na **Listagem 14**.

É definida na linha 03 uma variável do tipo TSMPrincipalClient, que nada mais é do que a representação do Server Module principal presente no servidor de aplicação. Na linha 05 criamos a classe Proxy passando como parâmetro a própria conexão por meio do método DBXConnection do sqlConexao. Em seguida na linha 07, o valor é atribuído ao campo IDCONTATO realizando a chamada do método getValorChave, passando o próprio nome do campo como parâmetro (da mesma forma que antes). Finalmente, após a obtenção do valor a classe é destruída e liberada da memória conforme a linha 09.

Para finalizar a última alteração é realizada no método de pesquisa do frmPesquisa. Conforme vimos anteriormente a pesquisa era realizada através da concatenação da própria SQL no formulário. Repare na **Listagem 15** a nova implementação.

Listagem 14. Utilizando a Proxy Client gerada pelo dbExpress

```
01 procedure TDMPrincipal.cdsAgendaNewRecord(DataSet: TDataSet);
02 var
03   SMPrincipal : TSMPrincipalClient;
04 begin
05   SMPrincipal := TSMPrincipalClient.Create(sqlConexao.DBXConnection);
06   try
07     cdsAgendaIDCONTATO.Value := SMPrincipal.getValorChave('IDCONTATO');
08   finally
09     FreeAndNil(SMPrincipal);
10   end;
11 end;
```

Listagem 15. Nova implementação do botão de pesquisa

```
01 procedure TfrmPesquisa.btnPesquisaClick(Sender: TObject);
02 begin
03   DMPrincipal.cdsPesquisa.Close;
04   DMPrincipal.cdsPesquisa.Data :=
       DMPrincipal.cdsPesquisa.DataRequest(
       VarArrayOf([cmbParametro.ItemIndex, edtPesquisa.Text]));
05 end;
```

Como pode-se notar, a implementação tornou-se muito mais simples do que antes. Na linha 03 o cdsPesquisa é fechado para que em seguida a propriedade .Data (que nada mais é do que a representação dos dados) receba o conteúdo do método DataRequest (linha 04). Este por sua vez aciona o evento OnDataRequest do DataSetProvider, montanto um Array através de VarArrayOf que permite passar inúmeros parâmetros na cadeia de arrays dinâmicos. Perceba que o primeiro conteúdo refere-se ao campo selecionado no ComboBox, o que no DataSetProvider é representado pelo Input[0], já o segundo parâmetro é o valor do conteúdo informado no Edit, que logicamente refere-se ao Input[1].

Se você ainda assim deseja utilizar aplicações multicamadas, e quer começar reduzindo o impacto de sua aplicação C-S, uma boa dica é procurar reparar as regras de negócio entre camadas diferentes como vimos aqui. Outra prática que pode ser adotada em ambiente C-S é utilizar um Data Module para concentrar seus ClientDataSets e outro para concentrar os SQLDataSets e DataSetProviders, deste modo a conexão entre eles pode ser realizada através do componente LocalConnection e das propriedades RemoteServer. Isto pode fazer com que seja possível migrá-los futuramente de uma maneira mais simples.

Conclusão

Como vimos aqui, as vantagens de se utilizar a arquitetura multicamadas em suas aplicações são inúmeras, e dependendo dos casos, realizar a migração pode não ser um processo tão complexo. O importante é ter consciência de que em casos de refatoração, este pode ser um fator a mais a ser considerado.

Durante o exemplo procuramos aplicar situações mais próximas da realidade dos desenvolvedores, permitindo assim uma visualização melhor dos problemas e realização das soluções de maneira mais rápida, adotando várias práticas comuns utilizadas em ambientes de produção com estas arquiteturas. Espero que este artigo seja utilizado como refêrencia para iniciar projetos neste modelo, tornando a vida de quem está iniciando, um pouco mais simples. Um grande abraço e até a próxima!

Daniel Sobrinho Laporte
daniel.laporte@gmail.com/ @daniel_laporte
Colunista e editor técnico da revista Clube Delphi. Graduado em Sistemas de informação e pós-graduado em administração de bancos de dados. Trabalha como desenvolvedor independente auxiliando na criação de aplicações customizadas e entrega de projetos. Desenvolve em Delphi desde 2005 adotando arquiteturas cliente-servidor e multicamadas com os mais variados bancos de dados.

Firebird
www.firebirdsql.org

Blogs do Autor
twitter.com/daniel_laporte

DevSpace do Autor
www.devmedia.com.br/daniel_laporte

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

InterBase XE – Parte 2

Conheça os componentes do Delphi relacionados a este SGBD

Este artigo faz parte de um curso

Resumo DevMan

De que se trata o artigo:

Esta segunda e última parte trata de exibir uma abordagem mais prática quanto ao acesso ao banco de dados InterBase XE. Para isso, são levantados alguns conceitos que irão complementar a apresentação do produto, ao mesmo tempo em que uma demonstração prática é realizada sobre os mesmos. A intenção fundamental é fazer com que o desenvolvedor Delphi se sinta à vontade no que diz respeito aos aspectos teóricos e práticos que envolvem este SGBD da mesma fabricante de seu IDE de desenvolvimento, o Delphi.

Em que situação o tema é útil:

Em muitos dos casos, é preciso também manipular esta base e ter o conhecimento suficiente para poder tirar proveito dos recursos que o SGBD lhe provê. Dito isto, esta série de artigos relacionados ao InterBase, se fundamentam na apresentação do produto de forma geral, a fim de torná-lo o mais familiar possível ao desenvolvedor Delphi.

InterBase XE – Parte 2 :

Como se sabe o InterBase é o SGBD provindo da mesma família do Delphi, mais especificamente da mesma fabricante, a Embarcadero. O produto que teve sua origem há mais de uma década atrás, ainda se apresenta de forma robusta ao mercado, mesmo após um período de ausência dos holofotes, em que muitos pensaram que o mesmo havia sido descontinuado. Prova disso é que muitos novos desenvolvedores tiveram seu início no Delphi já sem qualquer tipo de menção ao InterBase, desconhecendo assim sua existência. Por esses e outros motivos esta série de artigos se inspirou para trazer novamente à tona o assunto em meio à comunidade. Logo, além de um overview, o foco foi pontuar a mais recente versão do SGBD, a XE, e expor algumas das características e recursos do produto, muitas destas já provindas de suas versões anteriores. Em contraste à primeira parte desta série de artigos em que foram abordados as nuances mais conceituais do InterBase, nesta segunda e última parte será explorada a parte mais prática do contexto.

Na primeira parte desta série de artigos que trata do InterBase, foi apresentado todo um apanhado conceitual em relação ao produto, desde seu histórico até a amostragem de algumas de suas principais características, provindas de suas versões mais anteriores até a XE, sua versão mais atual. Contudo, a intenção primordial não foi apenas a de trazer novamente o assunto à tona para a comunidade, mas sim a de prover recursos suficientes de ordem conceitual e prática para que o desenvolvedor Delphi se sinta apto a desfrutar deste SGBD em suas aplicações. Portanto, sem perder tempo, será visto a seguir de que forma se dá a inserção do InterBase na IDE do Delphi, o que servirá de ponto de partida para sua utilização.

Componentes InterBase no Delphi

O fato de InterBase e Delphi serem da mesma fabricante tem como resultado o provimento de um conjunto de componentes específicos, localizados dentro do IDE de desenvolvimento, para se trabalhar com o SGBD. Tal conjunto apresenta-se sob o nome de InterBase Express, ou simplesmente IBX, como ficou popularmente conhecido em meio a comunidade. A finalidade destes componentes é de fornecer ao desenvolvedor um meio rápido e eficiente de se construir aplicações Delphi que possam efetivamente acessar, administrar e monitorar bancos de dados InterBase. Adicionalmente, permitir ainda a execução de serviços relacionados ao SGBD, em nível de servidor, a partir destas mesmas aplicações.

Olhando pelo lado mais prático do desenvolvimento Delphi relacionado a acesso a banco de dados, uma das grandes vantagens em se utilizar componentes específicos a uma tecnologia, está relacionada diretamente à questão de performance. Tomando como exemplo o próprio IBX, seu desempenho é altamente otimizado em razão de seu acesso direto à API do InterBase, fato este que o possibilita ainda tirar proveito de outros recursos inerentes ao SGBD, tais como seus serviços de Backup e Restore de banco de dados.

No Delphi XE2, os componentes IBX ficam localizados na Tool Palette do IDE, distribuídos em duas guias distintas, cada qual com uma finalidade diferente. Dessa forma, na guia InterBase (**Figura 1**), estão os componentes direcionados às tratativas de acesso a dados, enquanto que na guia InterBase Admin (**Figura 2**), estão os componentes para lidar com a parte administrativa do SGBD.

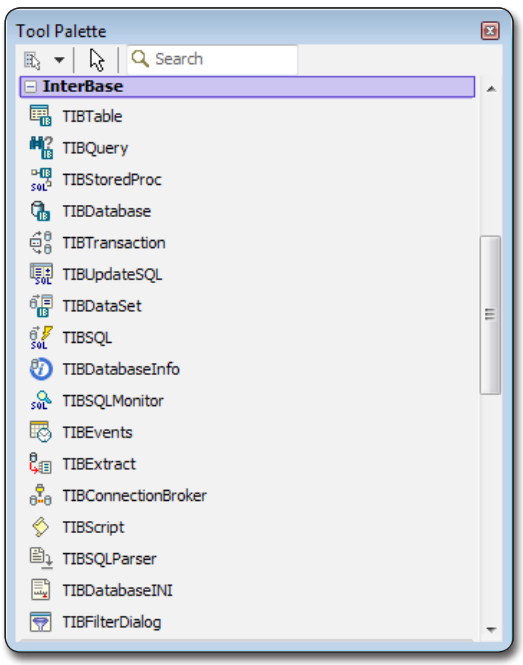

Figura 1. Guia de componentes InterBase na Tool Palette do Delphi XE2

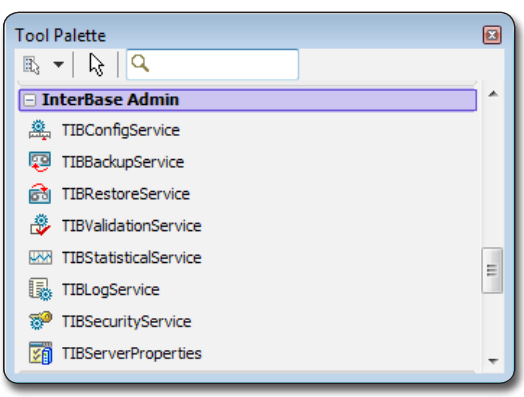

Figura 2. Guia de componentes InterBase Admin na Tool Palette do Delphi XE2

Características dos componentes

Conforme pôde ser visto, a gama de elementos relacionados ao InterBase disponíveis no IDE do Delphi é enorme. Em função disso, visando uma melhor contextualização do assunto, veremos a seguir a característica de alguns deles. A importância em se conhecer estes componentes um pouco mais de perto se dá pelo fato de que, será através de seu uso que o desenvolvedor Delphi poderá direcionar suas aplicações para este SGBD.

TIBTable

O componente TIBTable mantém um conjunto de dados provindos de um banco de dados InterBase, sem que se faça uso de qualquer instrução SQL. Dessa forma, ao invés de se informar o comando "Select" para retornar os dados de determinada tabela, por exemplo, você deverá apenas indicar o objeto em meio a uma lista de seleção provida pelo componente. Esta lista é refletida em sua propriedade TableName, conforme mostra a **Figura 3**. Em termos de comportamento, o TIBTable é considerado live, uma vez que se mantém conectado a determinado objeto do banco por todo o tempo. A origem dos dados varia então entre dois tipos de objetos, Tables ou Views, e é determinado pela propriedade TableTypes. No caso das tabelas, estas incluem além das tabelas de dados, as tabelas internas de sistema, conhecidas como System Tables. O trecho de código da **Listagem 1** mostra um componente TIBTable relacionado à tabela COUNTRY do banco de dados de exemplo Employee, que acompanha o InterBase. Além disso, ele está configurado para suportar também tabelas de sistema e Views (opções ttSystem e ttView).

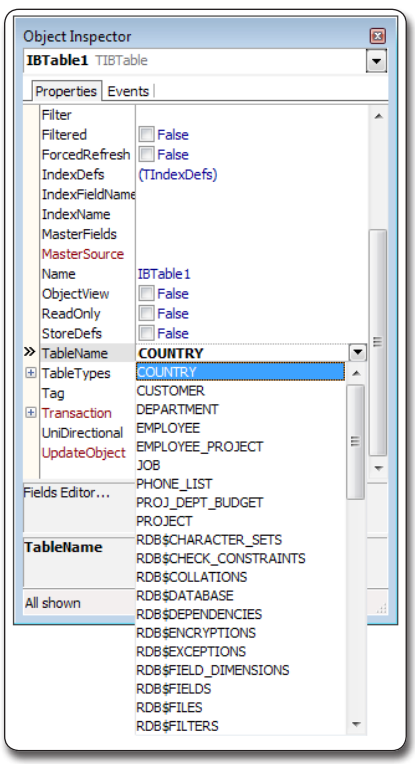

Figura 3. Lista de seleção da propriedade TableName de um componente TIBTable

Listagem 1. TIBTable relacionado à tabela COUNTRY

```
object IBTable1: TIBTable
  ...
  TableName = 'COUNTRY'
  TableTypes = [ttSystem, ttView]
end
```

TIBQuery

Ao contrário do TIBTable, TIBQuery é um componente típico para se executar instruções SQL dinâmicas sobre um banco de dados, incluindo comandos que vão de uma simples seleção até a inserção/atualização de dados. Sua utilização mais comum é, sem dúvida, direcionada à obtenção de dados do banco. Dessa forma, por se tratar de um componente para instruções dinâmicas você é capaz de, por exemplo, obter os dados de uma ou mais tabelas relacionadas, através de um comando SQL utilizando SELECT e JOIN. Dentre suas propriedades, "SQL" é a responsável por receber a instrução SQL, conforme mostra trecho de código da **Listagem 2**. Nesta declaração de um componente TIBQuery, podemos ver a exemplificação de uma obtenção de dados de duas tabelas, com uso de JOIN.

Listagem 2. Atribuição da instrução SQL à propriedade de mesmo nome

```
object IBQuery1: TIBQuery
  ...
  SQL.Strings = (
   'select EMPLOYEE.FIRST_NAME,'
   '       EMPLOYEE.LAST_NAME,'
   '       DEPARTMENT.DEPARTMENT'
   'from EMPLOYEE'
   'inner join DEPARTMENT'
   'on (DEPARTMENT.DEPT_NO =
   EMPLOYEE.DEPT_NO)')
end
```

Listagem 3. Stored Procedure ALL_LANGS

```
BEGIN
 FOR SELECT job_code, job_grade, job_country
 FROM job
  INTO :code, :grade, :country

 DO
 BEGIN
   FOR SELECT languages FROM show_langs
    (:code, :grade, :country) INTO :lang DO
     SUSPEND;
   code = '=====';
   grade = '=====';
   country = '==============';
   lang = '==============';
   SUSPEND;
 END
END
```

Listagem 4. Componente IBStoredProc

```
object IBStoredProc1: TIBStoredProc
  ...
  StoredProcName = 'ALL_LANGS'
  ...
end
```

TIBStoredProc

TIBStoredProc, como o próprio nome sugere, é um componente para procedimentos armazenados (Stored Procedures) de um banco de dados InterBase. Contudo, seu uso não diz respeito a todo e qualquer objeto deste tipo. Isto porque, ele se restringe a Procedures executáveis, cujo resultado retorne no máximo uma linha de informação. Sendo assim, uma Stored Procedure que retorne múltiplas linhas, tal como um procedimento de seleção (com uso de SELECT), está descartada, sendo recomendado seu uso com um componente TIBQuery. Este tipo de informação, aliás, é provida pelo próprio IDE, ao se tentar executar uma Stored Procedure não suportada. Para ilustrar este cenário e contribuir no entendimento, tomemos como exemplo o objeto Stored Procedure denominado ALL_LANGS, proveniente do banco de dados Employee do InterBase. Sua estrutura é então mostrada a seguir na **Listagem 3**.

Mediante o código e a teoria apresentada, podemos ver que este procedimento armazenado não é para ser usado com o componente TIBStoredProc, uma vez que se trata de uma seleção de dados cujo retorno terá múltiplas linhas como resultado. Supondo que não conhecêssemos o componente a fundo e nossa intenção seria a de utilizá-lo como um componente de dados comum, para armazenar os dados de retorno da Procedure do banco. Sendo assim, imediatamente vincularíamos o procedimento a um TIBStoredProc através de sua propriedade StoredProcName (**Listagem 4**).

A seguir, intuitivamente tentaríamos dar um Open no mesmo, da mesma forma que tradicionalmente fazemos com um componente TIBQuery. Ao executar a aplicação, no momento do Open, o Delphi levantaria uma exceção, conforme mostra a **Figura 4**. Essa exceção sugere explicitamente o uso de um componente TIBQuery para procedures de seleção de dados como esta. Sem deixar de mencionar que a execução de uma Stored Procedure "executável" através de um TIBStoredProc se dá através de seu método ExecProc, conforme também mencionado na mensagem.

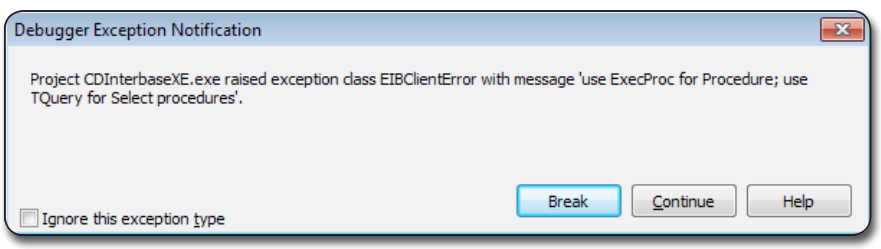

Figura 4. Exceção levantada ao se tentar executar um procedimento inválido ao componente

TIBDatabase

Componente essencial para se trabalhar com o IBX em aplicações Delphi, o TIBDatabase é o responsável por estabelecer a conexão com um banco de dados InterBase. Ao utilizá-lo, você pode lidar com uma ou várias transações simultâneas. No contexto do IBX uma transação é representada também por um componente, o TIBTransaction, que será tratado a seguir. Por padrão, apenas uma transação é definida a um TIBDatabase inicialmente através de sua propriedade DefaultTransaction. Contudo, você pode ter várias transações representadas por vários TIBTransaction, cada qual ligado ao mesmo componente de conexão, por exemplo. Ainda a respeito da conexão com o banco de dados propriamente dita, a **Figura 5** mostra o editor provido pelo TIBDatabase, utilizado para se configurar a conexão com um banco de dados InterBase. Sendo assim, é nesta tela que você irá informar alguns dados, tais como o tipo de conexão (se local ou remota), o caminho da base, nome do usuário e senha de acesso. Para aqueles que já estão habituados com o dbExpress, esta tela não chega a ser novidade, visto sua semelhança com seu editor de conexões.

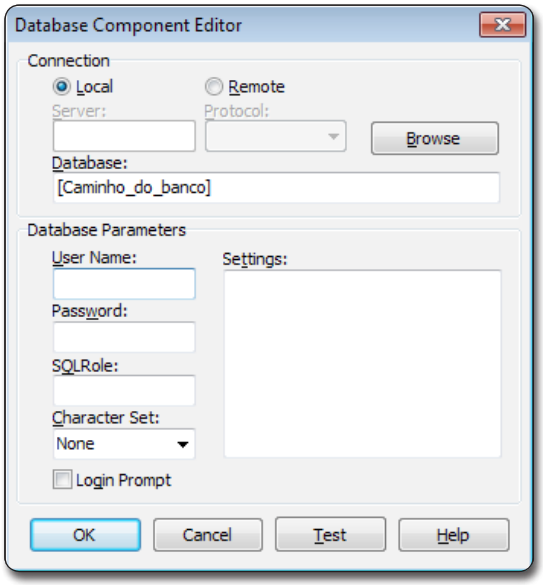

Figura 5. Editor de conexão do componente TIBDatabase

TIBTransaction

O TIBTransaction é o responsável por lidar com contextos de transações no cenário IBX, envolvendo para isso uma ou várias conexões com banco de dados. Apesar da citação de "várias conexões", o cenário mais comum encontrado em aplicações Delphi com InterBase Express, mesmo que não seja uma regra, são os com uma única conexão com banco, relacionada a uma ou múltiplas transações. A associação de um TIBDatabase no componente de transação se dá por uma propriedade chamada DefaultDatabase. StartTransaction, Commit, CommitRetaining, Rollback e Rollback Retaining são alguns dos métodos mais comuns de um TIBTransaction, a serem usados nas aplicações Delphi.

Na prática, o StartTransaction inicia efetivamente uma transação no contexto da aplicação e de sua conexão. Sendo assim, toda e qualquer ação subsequente, relacionada ao banco de dados da conexão, será considerada parte da operação até que a referida transação seja explicitamente encerrada, através do uso, por exemplo, do método Commit. Este irá então aplicar de forma permanente na base de dados todas as alterações realizadas durante a transação, encerrando-a a seguir. De forma similar, o CommitRetaining também efetiva as alterações, mas mantém a transação corrente ativa. Por fim, Rollback e Rollback Retaining descartam quaisquer alterações atreladas a uma transação, não as replicando sobre o banco de dados. Assim como ocorre com Commit e CommitRetaining, com o Rollback a transação é encerrada, enquanto que com o RollbackRetaining ela se mantém. Todos os métodos colaboram para se elevar o grau de importância com o uso de transações em meio ao desenvolvimento de Software, refletido no conceito de ACID (ver **Nota do DevMan 1**).

TIBSQL

O TIBSQL pode ser considerado um componente para operações rápidas envolvendo SQL. Sendo assim, ele é comumente utilizado para executar eventuais comandos sobre dados, tais como INSERT, UPDATE e DELETE, além de operações DDL, para definições de dados. O trecho de código da **Listagem 5** mostra o componente configurado com uma instrução DDL, que irá criar uma nova tabela no banco de dados. No geral, para sua utilização é necessário o preenchimento de três de suas propriedades - Database, Transaction e SQL - que irão apontar para um componente de conexão (TIBDatabase), um componente de transação (TIBTransaction) e a instrução SQL respectivamente. Seu método no qual executará a instrução na base de dados é denominado ExecQuery.

Listagem 5. Configuração do TIBSQL

```
object IBSQL1: TIBSQL
  Database = IBDatabase1
  Transaction = IBTransaction1
  SQL.Strings = (
   'CREATE TABLE TABELA_TESTE ('
   '   CODIGO INTEGER NOT NULL,'
   '   DESCRICAO VARCHAR(80) NOT NULL)')
  ...
end
```

Nota do DevMan 1

ACID (acrônimo de Atomicidade, Consistência, Isolamento e Durabilidade) é um conceito computacional utilizado para caracterizar uma transação envolvida em dada aplicação com banco de dados. A atomicidade visa garantir que todas as operações atreladas a uma transação sejam devidamente executadas, caracterizando assim o sucesso do processo. Caso ocorra alguma falha em alguma das operações, o processo como um todo é caracterizado falho, fazendo com que a base de dados participante não lide com resultados parciais de transação. A consistência prevê que os dados envolvidos numa transação voltem ao seu estado inicial, no caso de falha de alguma das operações da mesma. Já o isolamento diz respeito a uma forma de garantir que apenas uma entidade acesse o estado dos dados que estão sendo manipulados no momento da transação corrente, evitando que outras entidades tenham acesso a estados e valores que não sejam finais. Por fim, a durabilidade certifica que todos os dados estarão disponíveis de forma definitiva, uma vez que a transação é efetivada.

TIBSQLMonitor

O TIBSQLMonitor é o componente IBX para diagnóstico e monitoramento da comunicação entre a aplicação Delphi e o servidor de banco de dados InterBase, coletando informações sobre as instruções SQL que são transmitidas. Dessa forma, torna-se possível, por exemplo, a elaboração de ferramentas para uso interno que irão fazer o acompanhamento de todas as atividades relacionadas à conexão com um determinado servidor InterBase, direcionando um log de informações para um arquivo externo, ou mesmo para um componente de texto da VCL. Isso significa dizer que através da elaboração de uma única ferramenta que faça uso de um TIBSQLMonitor, você é capaz de monitorar outras aplicações IBX que façam conexão ao mesmo servidor. A **Figura 6** exibe então uma situação onde o Log foi direcionado a um TMemo para visualização em tela das informações. No caso, o componente de monitoramento capturou três erros ocorridos durante a execução de uma segunda aplicação, contendo erros de SQL.

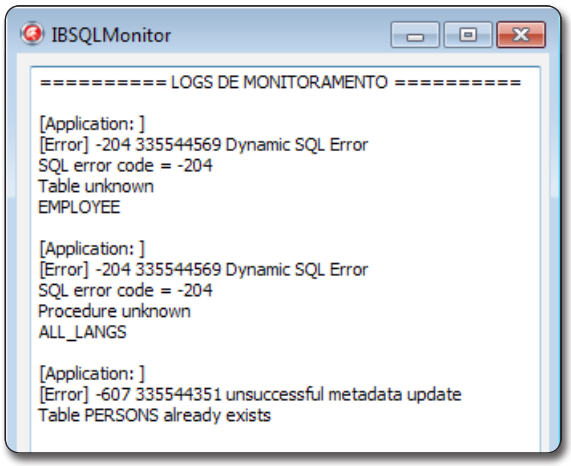

Figura 6. Informações obtidas pelo TIBSQLMonitor sendo direcionadas a um Memo

Na prática, para se acessar as informações de monitoramento do componente, é utilizado seu principal e único evento, que é denominado OnSQL. Sua declaração pode então ser vista a seguir, onde nota-se a presença de dois parâmetros, EventText, que se refere ao texto das informações de monitoramento, e EventTime, que é a data e hora de seu acontecimento.

```
procedure TForm1.IBSQLMonitor1SQL(EventText: string; EventTime: TDateTime);
```

TIBBackupService

Deste ponto em diante serão vistos alguns dos componentes IBX referentes a serviços. Dito isto, o TIBBackupService é o componente responsável por fazer o backup de um banco de dados InterBase. Ele provê então toda a gama de opções de backup (database backup options), sendo uma alternativa visual e eficiente ao gbak, o utilitário de linha de comando para backup/restore que acompanha a instalação do InterBase.

Vendo um pouco de seu uso na prática, iremos elucidar a seguir uma aplicação Delphi que, através do uso do componente, terá a incumbência de realizar o backup da base de dados employee do InterBase em um servidor local. A **Listagem 6** mostra então um exemplo de código utilizado para se configurar o TIBBackup Service dinamicamente para, posteriormente, dar início ao processo propriamente dito.

Tendo em vista o código apresentado, é importante se fazer algumas observações. Das linhas 4 até a 8 são preenchidas as propriedades do componente referente a nome do servidor a ser conectado e as credenciais de acesso (usuário e senha). Para se efetivar a conexão, o componente é então ativado (Active = True). As linhas 10 e 11 definem os valores para as propriedades DatabaseName e BackupFile relativas ao caminho do banco de dados que será feito o backup e o caminho do arquivo de backup a ser gerado, respectivamente. Por fim, na linha 13 é chamado o método ServiceStart, que irá efetivamente dar início ao processo.

De forma proposital, os comentários sobre a propriedade Options, declarada na linha 12 foram deixados para o fim, em virtude do conceito envolvido necessitar de uma explicação maior. Pelo que sua própria nomenclatura sugere, tal propriedade se refere às opções de backup, ou "database backup options", citadas anteriormente. Somando um total de sete, cada opção apresenta-se com valor booleano, podendo então estar "ativada" ou não. A **Figura 7** mostra em detalhes as opções providas pelo componente, em sua propriedade Options.

Listagem 6. Exemplo de código para uso do componente TIBBackupService

```
01 with IBBackupService1 do
02  begin
03  try
04    LoginPrompt := False;
05    ServerName := 'localhost';
06    Params.Add('user_name=sysdba');
07    Params.Add('password=masterkey');
08    Active := True;
09
10    DatabaseName := '[caminho]\employee.gdb';
11    BackupFile.Add('[caminho]\employee_.gbk');
12    Options := [];
13    ServiceStart;
14  finally
15     Active := False;
16  end;
17 end;
```

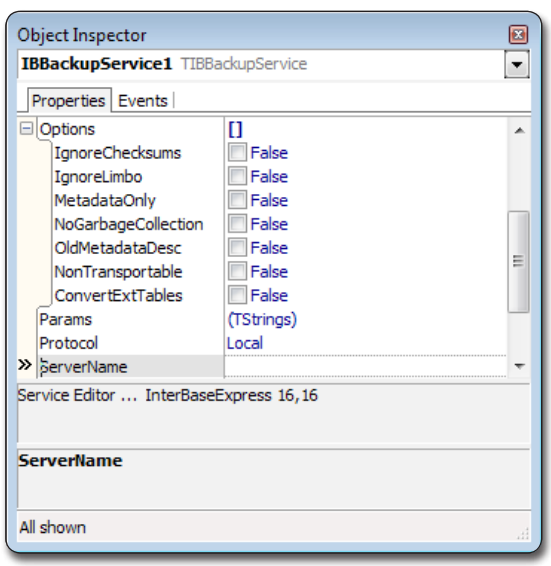

Figura 7. Backup Options no componente TIBBackupService

Sem deixar de mencionar, a seguir é apresentada uma breve descrição sobre algumas destas principais opções:

- MetadataOnly: faz com que o backup gerado contenha somente as definições dos objetos do banco, ou seja, os metadados. Os dados não são envolvidos no processo, resultando em tabelas vazias no arquivo de backup gerado;
- NoGarbageCollection: com a intenção de melhorar a performance do processo, descarta-se o recurso de garbage collection durante o backup;
- OldMetadataDesc: pré-formata os metadados de saída;

• NonTransportable: faz com que o arquivo de saída de backup fique num formato de dados "non-XDR", com a intenção única de uma busca pela melhora de espaço e desempenho;
• ConvertExtTables: converte dados de tabelas externas para tabelas internas.

TIBRestoreService

O TIBRestoreService pode ser considerado, em muitos dos casos, como um complemento ao componente TIBBackupService, uma vez que lida com o processo de restauração (Restore) de uma base de dados InterBase. Diz-se complemento pelo fato de um Restore somente ocorrer mediante um arquivo de backup existente. Através de seus métodos e propriedades, o componente disponibiliza tudo o que é pertinente ao processo, tais como os parâmetros e as opções de Restore (Database Restore Options) suportadas pelo SGBD.

Da mesma forma que foi feito anteriormente, será exposto agora um exemplo de caso, onde poderemos ver seu uso em termos de código e configuração. Sendo assim, ao se arrastar um componente TIBRestoreService para ser utilizado em uma aplicação Delphi, o preenchimento de algumas de suas propriedades acabam por se tornar quase que obrigatórias, dada a importância de cada uma ao processo. A primeira delas é a denominada PageBuffers, que está relacionada ao tamanho do cache do banco de dados. Sendo mais específico, é ela quem irá determinar o número de páginas de buffer no cache do banco de dados restaurado. Na mesma medida, PageSize é outra propriedade importante, e que diz respeito ao tamanho de página do banco de dados. Seu valor é determinado em bytes, num intervalo que varia entre 1024 a 16384. Por fim, estão as opções de Restore que de forma semelhante às opções de backup, são apresentadas sob a forma de itens booleanos. Sendo assim, as seguintes "database restore options" estão disponíveis: DeactivateIndex, NoShadow, NoValidity, OneRelationATime, Replace, CreateNewDB, UseAllSpace e ValidationCheck. Dentre as mais usuais estão Replace, que faz com que o banco de dados restaurado substitua um existente, CreateNewDB, que irá fazer com que o banco restaurado não substitua um existente e DeactivateIndex, que não constrói qualquer tipo de índice durante o processo de restauração.

A partir disso, uma configuração básica para o componente pode ser vista na **Listagem 7,** na qual refere-se à declaração de um TIBRestoreService no arquivo .dfm.

Finalmente, para se iniciar o processo de Restore através de componente, é utilizado seu método ServiceStart, da mesma forma que ocorre com TIBBackupService.

TIBValidationService

TIBValidationService é o componente da guia Interbase Admin designado para validar um banco de dados InterBase e reconciliar suas transações. Provido de oito opções relacionadas à "database validation", o componente expressa através de cada uma destas opções suas diferentes funcionalidades. Na declaração da classe TIBValidationService, presente na unit IBServices, podemos ver então a presença de sua propriedade Options, sendo do tipo TValidateOptions, conforme mostra o trecho da **Listagem 8**.

Listagem 7. Dfm do TIBRestoreService

```
object IBRestoreService1: TIBRestoreService
  Active = True
  ServerName = 'localhost'
  Protocol = TCP
  Params.Strings = (
   'user_name=sysdba'
   'password=masterkey')
  LoginPrompt = False
  TraceFlags = []
  ServerType = 'IBServer'
  DatabaseName.Strings = (
   '[caminho_do_banco_restaurado]')
  BackupFile.Strings = (
   '[caminho_do_arquivo_de_backup]')
  PageSize = 8192
  PageBuffers = 2048
  ...
 end
```

Listagem 8. Propriedade Options do tipo TValidateOptions

```
TIBValidationService = class(TIBControlAndQueryService)
private
  ....
  FOptions: TValidateOptions;
  ...
published
  ...
  property Options: TValidateOptions read FOptions write FOptions;
```

Ainda inspecionando a referida unit, vemos que tal tipo (TValidateOptions) na verdade se trata de um conjunto de elementos do tipo TValidateOption, que se refere a um enumerado de opções. Tanto a declaração de TValidateOptions, quanto de TValidateOption são expressas a seguir, exatamente como aparecem na unit.

```
TValidateOption = (LimboTransactions, CheckDB, IgnoreChecksum, KillShadows,
MendDB, SweepDB, ValidateDB, ValidateFull);

TValidateOptions = set of TValidateOption;
```

A seguir, são descritas algumas das principais opções de validação disponíveis:
• **LimboTransactions**: obtém o retorno das informações relacionadas a transações em limbo, tais como seu ID (Transaction ID), seu estado (Transaction state), sua ação (Transaction action) e o caminho do banco de dados remoto;
• **CheckDB**: solicita uma validação somente leitura (read-only) do banco de dados InterBase, de caráter apenas informativo, sem corrigir eventuais problemas detectados;
• **MendDB**: faz a marcação de registros corrompidos do banco de dados, a fim de ignorá-los em operações posteriores. Do contrário, tais registros podem comprometer todo um processo;

• **SweepDB**: solicita uma varredura do banco de dados, com o intuito de identificar possíveis registros desatualizados e marcá-los como "espaço livre";
• **ValidateDB**: localiza páginas do banco que não estejam atreladas a qualquer estrutura de dados, mas que por algum motivo estão indevidamente alocadas;
• **ValidateFull**: usado em conjunto do ValidateDB, localiza fragmentos não utilizados em estruturas de página e de registro, liberando-os a seguir;

TIBLogService

TIBLogService é um componente simplista para a criação de arquivos de log InterBase. Um servidor InterBase, por padrão, apresenta um arquivo de log denominado "interbase.log", localizado em seu diretório de instalação. As informações presentes neste arquivo se referem então às atividades inerentes ao Server, tais como o momento de seu "Start", ou algum eventual erro ocorrido durante sua execução. Sendo assim, este arquivo é atualizado a todo o momento, conforme as informações vão sendo geradas. Uma vez que este arquivo se hospeda no servidor, em um ambiente distribuído, o acesso a ele pode se tornar dificultado, o que dá margens para a utilização do componente em questão, por meio de uma aplicação cliente que irá funcionar como uma ferramenta. Em outras palavras, através de seu uso no Delphi, você poderá construir um utilitário que irá então recuperar as informações de log deste arquivo, trazendo-as para serem exibidas na aplicação.

Exemplificando este cenário, temos na **Figura 8** um formulário em modo Design contendo apenas três componentes (TButton, TMemo e TIBLogService), sugestivo à construção de um utilitário deste tipo.

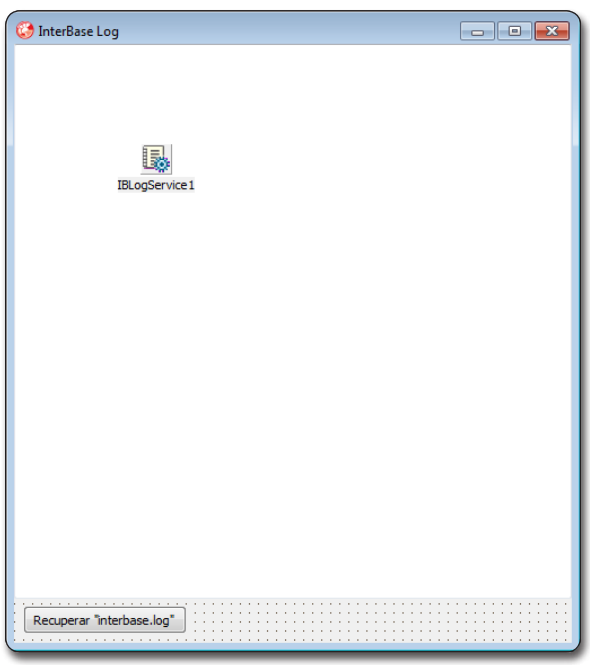

Figura 8. TIBLogService - Formulário de exemplo em modo Design

A seguir, a **Listagem 9** exibe a codificação do evento OnClick do TButton, necessária para a execução do processo. Neste mesmo momento é que é feita também a configuração dinâmica do componente TIBLogService.

Listagem 9. Codificação do evento OnClick do botão

```
01 procedure TForm1.Button1Click(Sender: TObject);
02 begin
03   with IBLogService1 do
04   begin
05     ServerName := 'localhost';
06     LoginPrompt := False;
07     Params.Add('user_name=sysdba');
08     Params.Add('password=masterkey');
09     Active := True;
10     ServiceStart;
11     while not Eof do
12       Memo1.Lines.Add(GetNextLine);
13   end;
14 end;
```

Neste código apresentado, das linhas 05 a 08 são configuradas basicamente as propriedades do componente relativas ao servidor que está se conectando. Para este exemplo, se supõe o trabalho com um servidor local, com as credenciais padrão de acesso (usuário: sysdba/ senha: masterkey). Na linha 09 o componente é ativado, estabelecendo assim a conexão com o servidor indicado. Uma vez conectado, é feita uma chamada ao método Service Start do componente, para que o processo seja iniciado. Por fim, nas linhas 11 e 12 são feitas as tratativas para listar no TMemo (Memo1) da aplicação cada linha de informação obtida do arquivo de log do InterBase.

A **Figura 9** mostra o aplicativo de exemplo em execução, listando as informações de log do arquivo "interbase.log" naquele momento.

dbExpress e InterBase

É importante ressaltar que, nativamente, o trabalho com InterBase no desenvolvimento Delphi não se restringe aos componentes IBX. Isto porque, o SGBD também é devidamente suportado pelo dbExpress, já muito conhecido da comunidade Delphi. Ele que é o Framework para se trabalhar com banco de dados de variados tipos, inclui o InterBase entre suas opções nativas, provendo um driver específico, denominado dbxint.dll. Driver este, aliás, que foi exaustivamente utilizado em aplicações Delphi que utilizavam dbExpress para se conectar a bancos de dados Firebird, tempos antes do lançamento do driver DBX específico para esta tecnologia.

O jeito mais trivial de se obter uma conexão com uma base de dados InterBase através do uso de dbExpress no IDE do Delphi é através da guia Data Explorer. Em sua estrutura, é disponibilizado então um nó específico para bancos deste SGBD, conforme mostra a **Figura 10**. Uma vez que uma nova conexão é registrada no Data Explorer, ela já está apta a ser utilizada em aplicações, pelo simples arrastar do nó da cone-

xão para um formulário ou Data Module. O próprio Delphi se encarrega então de adicionar ao projeto um novo componente TSQLConnection, já configurado à conexão.

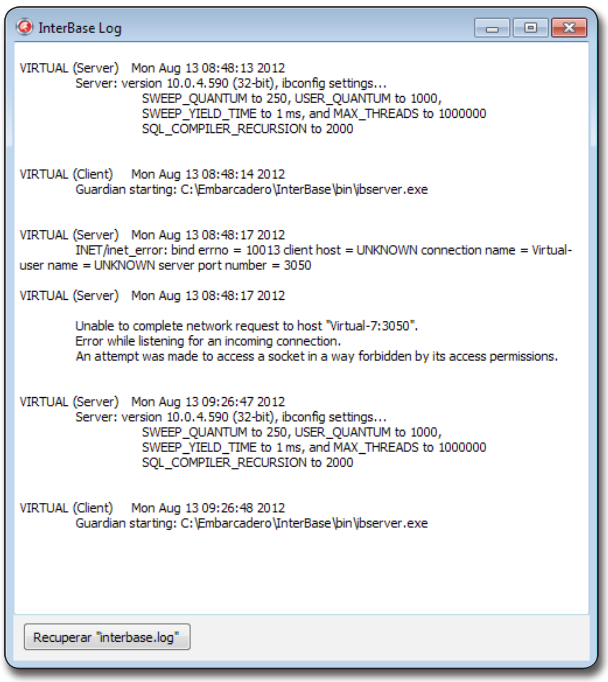

Figura 9. TIBLogService – Aplicação de exemplo em execução

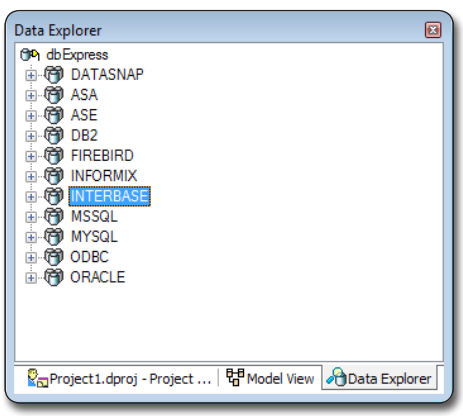

Figura 10. Nó InterBase no Data Explorer

Conclusão

Em termos práticos, a utilização do InterBase se assemelha muito ao Firebird, o que para o desenvolvedor Delphi torna-se uma coincidência agradável. Isto porque, o Firebird, sem dúvida nenhuma, é um dos SGBDs mais utilizados pela comunidade em projetos Win32. Como você pôde ver na primeira parte desta série de artigos, tal semelhança não é por caso, visto que o InterBase serviu como base para o desenvolvimento do Firebird. Ainda remontando a história, foi visto de forma abrangente todo o processo evolutivo do InterBase, até o seu momento atual em sua versão XE.

Soando como uma novidade para alguns, foi mostrado também que o InterBase XE traz dentre suas edições uma de caráter totalmente gratuita denominada Developer Edition. Apesar de algumas restrições de caráter técnico, esta distribuição do produto é ideal e suficiente para o profissional Delphi se ambientar, testar e produzir aplicações com o InterBase. Em se tratando de desenvolvimento, esta última parte da série apresentou, entre outras coisas, os detalhes dos componentes presentes dentro do IDE do Delphi relacionados ao InterBase.

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br
Formado em Processamento de Dados pela FATEC-TQ. Atua como Analista Programador Delphi há 6 anos. Atualmente escreve artigos sobre Delphi para revistas especializadas.
Blog: prismsilver.blogspot.com

Embarcadero InterBase XE – Página oficial do produto
http://embarcadero.com/products/interbase

Embarcadero InterBase XE – Download
https://downloads.embarcadero.com/free/interbase

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Aplicando Segurança

Liberando e restringindo acessos

Segurança é um fator extremamente importante e muitas vezes indispensável em vários setores da vida, seja no trabalho ou no dia-a-dia, e porque não também nas informações, tornando-as restritas e protegidas, garantindo a integridade das mesmas. Muitos sites presentes na Internet, bem como WebApps, são um exemplo prático dentro deste contexto, onde muitos mecanismos são empregados com o objetivo de reduzir ao máximo o número de fraudes e acessos indevidos, bem como preservar os dados transferidos e as informações de privacidade dos usuários, assim como a autenticidade dos mesmos através da utilização de protocolos específicos, criptografia, certificados, logins, senhas, entre outros recursos presentes na atualidade.

Garantir que determinados dados apenas serão gravados e acessados por pessoas específicas também é um diferencial necessário em muitas aplicações comerciais e industriais, sejam elas de pequeno, médio ou grande porte. Em softwares do tipo ERP (ver **nota do DevMan 1**), por exemplo, que normalmente são compostos por vários módulos acessados por diversos usuáriosa todoinstante, o emprego de segurança e limitações a áreas específicas se torna um fator obrigatório, afinal, usuários ou grupos de um determinado setor, com exceção de administradores e alguns departamentos, não necessitam acessar ou alterar os dados que pertencem a outro, por questões de inconsistências e até mesmo éticas, uma vez que tal função já está atribuída e é de responsabilidade de outro setor.

Quanto ao que concerne à elaboração de softwares capacitados a empregar conceitos de segurança, os desenvolvedores contam com uma variedade de alternativas, que vão desde implementações manuais, que consequentemente demandam maior tempo e esforços, tais como a criação tabelas específicas no banco de dados para armazenar informações de acesso dos usuários e funções no código fonte para a autenticação e validação dos mesmos, até componentes e ferramentas completas presentes no mercado, como é o caso do UserControl e do TMS Security System, disponíveis para o Delphi XE 2 etambém para versões anteriores.

Em síntese, o UserControl é uma ferramenta de muita utilidade onde o desenvolvedor utiliza componentes e janelas específicas do mesmo de uma forma rápida e prática, em tempo de design, para adicionar os componentes (menus, botões, etc) que serão controlados, possuindo também componentes de integração com os frameworks de bancos de dados, aumentando consideravelmente a produtividade no que diz

Resumo DevMan

De que se trata o artigo:

Este artigo apresenta algumas formas para a implementação de segurança nas aplicações através da autenticação dos usuários e o carregamento das permissões para os mesmos, demonstrando primeiramente uma solução desenvolvida manualmente e posteriormente, através do componente UserControl, facilitando e automatizando o emprego deste conceito e recurso. Também é efetuada uma introdução ao componente TMS Security System, outra ferramenta que tem esta finalidade.

Em que situação o tema é útil:

Aumentar as funcionalidades das aplicações para o usuário final, permitindo que o mesmo, através de uma conta de administrador, possa criar outros usuários e definir quais serão as permissões de acesso de cada utilizador do sistema, visando preservar e garantir a integridade e sigilo de determinadas informações.

Segurança :

Segurança da informação, sem dúvida, é um fator indispensável para as aplicações, principalmente para aquelas que são utilizadas por diversas pessoas e oferecem vários serviços de forma centralizada, abrigando toda a regra de negócio da empresa. Sendo assim, quando o assunto é desenvolvimento de software, convém a implementação de mecanismos que possibilitem uma definição de privilégios para seus utilizadores, atribuindo áreas e opções que os mesmos podem utilizar ou são responsáveis e bloqueando aquelas que eles não devem ter acesso por diversas questões, isso para preservar a integridade e o sigilo das informações. Entretanto, segurança é um fator que também deve envolver todo o ambiente que está relacionado ao software, visando assim, garantir ao máximo a privacidade dos dados. Neste artigo é apresentada uma forma manual de empregar um controle de acesso para permitir ou bloquear as funcionalidades do sistema, através da autenticação dos usuários durante a inicialização do mesmo, em seguida é demonstrado um exemplo com o componente UserControl, que pode ser empregado dentro deste contexto, automatizando e facilitando muito a implementação deste recurso nos projetos e para concluir, é efetuada uma introdução ao conjunto de componentes do TMS Security System.

respeito à implementação de segurança. Sendo assim, através dos componentes desta ferramenta, um usuário administrador poderá criar ou remover usuários, liberar ou restringir seus acessos, entre outros recursos,em tempo de execução.

Já o TMS Security System, desenvolvido pela TMS Software, também é uma ferramenta que suporta a atribuição de permissões de acesso aos usuários através de uma conta de administrador, permitindo as definições em níveis de formulário e menus, que por sua vez podem estar agrupados e herdarem os direitos. Contudo, os desenvolvedores ainda contam com outros recursos, como autenticação integrada através de usuários do Windows.

É possível notar que a funcionalidade de segurança e restrição das informações é algo importante e útil para as aplicações, atribuindo o que cada usuário do sistema pode acessar e gravar, mas esta forma de proteção está limitada ao executável do mesmo, o que implica que outras boas práticas devem ser consideradas para aumentar o nível de segurança e integridade dos dados. Sendo assim, outros pontos fundamentais devem ser enfatizados, ou seja, conceitos como isolamento da base de dados em um servidor devidamente protegido, alteração da senha padrão do usuário administrador do software SGBD (Sistema de Gerenciamento de Banco de Dados), emprego de criptografia, backups regulares (cópias de segurança), entre outros, também devem ser levados em consideração e empregados em conjunto com a aplicação, afinal determinado usuário que tenha um maior conhecimento e experiência, pode facilmente acessar a base de dados e assim comprometer e efetuar a ação que desejar por meio desta.

Implementando Segurança na prática

Para iniciar uma aplicação de exemplo com o emprego de segurança, um novo projeto deve ser criado (File>New>VCL Forms Application), onde o formulário principal recebeu a denominação de FrPrincipal. Neste primeiro exemplo o controle de usuários e acessos será desenvolvido de uma forma manual, o que demanda maior tempo e consequentemente, mais código fonte. Sendo assim as informações serão gravadas em tabelas do banco de dados, onde o SGBD adotado foi o Firebird. A **Listagem 1** apresenta o código SQL para a criação das tabelas que serão utilizadas para a segurança da aplicação, sendo a primeira (USUARIO) para armazenar as informações dos usuários e a segunda (USUARIOACESSO), para seus devidos acessos. Crie um novo banco de dados e execute as sentenças SQL da **Listagem 1**.

É possível notar que a tabela de usuários contém um simples cadastro das informações mais necessárias, possuindo um campo mais específico que foi denominado "ADMINISTRADOR", com o objetivo de definir tal usuário com acesso total. Quanto à tabela de acessos, além da chave primária e do ID do usuário, existe um campo importante que recebeu a denominação de "COMPONENTE", cujo intuito é armazenar o nome do componente que foi definido no Delphi para representar quais os objetos que o usuário vinculado possui acesso, levando em conta também, o valor do campo "ATIVO", que por sua vez deve ser "S".

Com as tabelas devidamente criadas, a próxima etapa é adicionar um módulo ao projeto (File>New>Other>Data Module) com o intuito de centralizar os componentes em comum que serão utilizados em todos os formulários. Tal módulo recebeu a denominação de "dm". Sendo assim, foi inserido o componente responsável pela conexão com o banco de dados, o TSQLConnection do framework DBExpress, que foi adotado para este exemplo. Algumas propriedades de tal objeto receberam as configurações de acordo com o trecho a seguir:

```
ConnectionName = FBConnection
LoginPrompt = False
Params.Database = [Endereço do banco]
Connected = True
```

Também foram criadas algumas variáveis no módulo adicionado anteriormente com o intuito de armazenar as futuras informações de login, lembrando que as mesmas devem ser inseridas na seção

Nota do DevMan 1

ERP (Enterprise Resource Planning – Planejamento de Recursos Empresariais), também conhecido como Sistema de Gestão Empresarial, é uma denominação atribuída a um software que integra todos os processos de negócio deuma empresa. Normalmenteéum sistema desenvolvido com o intuito de atender os mais diversos nichos de mercado e é composto por diversos módulos destinados a cada departamento ou setor da organização, sendo amplamente utilizadono meio industrial. O emprego de um ERP traz diversas vantagens, tais como a centralizaçãoe sincronismo das informações, permitindo um acesso rápido aos dados de qualquer área,a redução de falhas, a diminuição dos custos, a minimização dos níveis de estoque,o aumento da eficiência em diversos segmentos, dentre outros. Entretanto, o processo de implantação de um ERP pode demandar certo tempo e consumir recursos financeiros, entre outros, uma vez que o mesmo atinge todas as áreas da empresa, todos os processos devem ser devidamente mapeados e as parametrizações definidas, além evidentemente, do treinamento dos colaboradores.

Listagem 1.Código SQL para a geração das tabelas

```
CREATE TABLE USUARIO(
   ID INTEGER NOT NULL,
NOME VARCHAR(50),
   LOGIN VARCHAR(20) UNIQUE,
   SENHA VARCHAR(20),
ADMINISTRADOR VARCHAR(1),
   ATIVO VARCHAR(1),
   CONSTRAINT PKY_USUARIO PRIMARY KEY(ID)
);
CREATE TABLE USUARIOACESSO(
   ID INTEGER NOT NULL,
   IDUSUARIO INTEGER NOT NULL,
   COMPONENTE VARCHAR(50),
   ATIVO VARCHAR(1),
   CONSTRAINT PKY_USUARIOACESSO PRIMARY KEY(ID),
   CONSTRAINT FKY_USUACE_USUARIO FOREIGN KEY(IDUSUARIO)
   REFERENCES USUARIO(ID)
);
```

var do módulo "dm" para que sejam públicas, ou seja, para que possam ser acessadas por outras partes do projeto. Tais variáveis foram declaradas de acordo com o trecho a seguir:

```
UsuarioLogado, UsuarioAdministrador: Boolean;
UsuarioID: Integer;
UsuarioAtual: string;
```

Para a interligação com as tabelas do banco de dados, foram adicionados dois componentes do tipo TSimpleDataSet e um TDataSource. O TSimpleDataSet, presente na aba dbExpress, é um componente deste framework que provê as funcionalidades necessárias para que as informações possam ser inseridas, obtidas, gravadas e excluídas no banco de dados facilmente (CRUD). Os objetos TSimpleDataSet adicionados receberam a denominação de sdsUsuario e sdsUsuarioAcesso. Na propriedade SQLConnection de ambos foi atribuído o valor de SQLConnection1, o componente responsável pela conexão.A propriedade DataSet.CommandText recebeu o código SQL equivalente às tabelas que os mesmos estão vinculados, onde por exemplo o comando definido para o sdsUsuario foi "SELECT * FROM USUARIO". Após a atribuição dos códigos SQL deve-se efetuar um clique duplo sobre tais objetos e adicionar os campos da tabela, através da opção "Addallfields", presente no menu de contexto de ambos. Isso irá criar os chamados campos persistentes, objetos que representam os campos da tabela. O objeto TDataSource, denominado dsUsuario, foi interligado ao sdsUsuario através de sua propriedade DataSet. Também foi utilizado o conceito de Master/Detail para vincular o sdsUsuario ao sdsUsuarioAcesso. Este processo pode ser efetuado configurando a propriedade MasterSource do objeto sdsUsuarioAcesso com o valor de dsUsuario e em seguida, definir os campos MasterFields com o valor de "ID" e IndexFieldNames com "IDUSUARIO", criando assim o vínculo com a tabela USUARIO. Para concluir esta etapa, basta ativar a propriedade Active de ambos os TSimpleDataSet, iniciando assim a interligação com suas respectivas tabelas do banco de dados.

Contudo, para que o sdsUsuarioAcesso possa efetivar as futuras alterações que serão efetuadas na tabela USUARIOACESSO através da aplicação, os eventos AfterDelete e AfterPost do mesmo receberam o comando a seguir:

```
sdsUsuarioAcesso.ApplyUpdates(0);
```

O ApplyUpdates efetua o envio das informações ao banco de dados. O parâmetro solicitado indica a quantidade de erros aceitável nessa operação de salvamento, no caso nenhum erro é aceitável. Assim, se algo acontece de errado, toda transação é desfeita e nada é salvo. Para concluir a configuração do módulo "dm", foi definida uma função com o intuito de retornar um código sequencial toda vez que um novo registro for criado no objeto sdsUsuarioAcesso. A **Listagem 2** apresenta o código da função responsável pela ação citada, que por sua vez, recebeu a denominação de GetNovoCodigoAcesso().

Listagem 2. Código da função GetNovoCodigoAcesso()

```
function Tdm.GetNovoCodigoAcesso: Integer;
var
  Qry: TSQLQuery;
begin
  Qry := TSQLQuery.Create(nil);
try
with Qry do
begin
      SQLConnection := SQLConnection1;
      SQL.Text := ' SELECT MAX(ID) FROM USUARIOACESSO ';
      Open;
      Result := Fields[0].AsInteger +1;
      Close;
end;
finally
    FreeAndNil(Qry);
end;
end;
```

Como o framework adotado para este projeto foi o DBExpress, foi declarada uma varável do tipo TSQLQuery (Qry), uma das classes responsáveis pela implementação de Querys (consultas) deste framework, para efetuar a devida pesquisa na tabela USUARIO ACESSO. A propriedade SQLConnection da mesma recebeu o valor do objeto responsável pela conexão com o banco, SQLConnection1 e sua propriedade SQL.Text, que deve conter a instrução SQL que será executada, recebeu o comando SELECT MAX() com intuito de obter maior número de ID presente em tal tabela, onde posteriormente o resultado da função recebe o valor da adição (MAX(ID) +1) para gerar um novo código sequencial. Para finalizar, a conexão é encerrada e o objeto é liberado da memória.

Concluído o módulo dm, a próxima etapa no projeto é inserir um novo formulário (File>New>VCL Form), para a elaboração da tela de login, que foi denominado FrLogin. Neste formulário foram inseridos um TGroupBox, dois TLabels, dois TEdits e um botão. A **Figura 1** apresenta o visual final do formulário FrLogin deste exemplo.

Com o visual do formulário de Login concluído, resta apenas adicionar o evento que será responsável por validar os dados digitados pelo usuário, sendo o mesmo atribuído ao evento OnClick do TSpeedButton adicionado. A **Listagem 3** apresenta o código do evento OnClick do btEfetuarLogin.

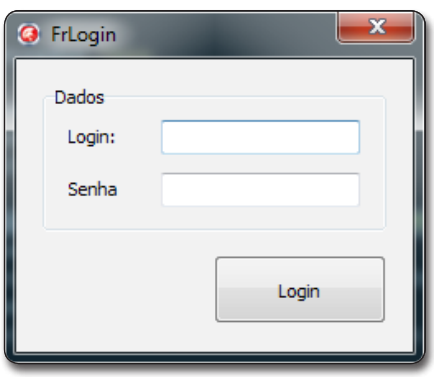

Figura 1. Visual final do formulário de Login

Nota

Os controles foram renomeados de acordo com sua funcionalidade no formulário

Listagem 3. Código do evento OnClick do botão btEfetuarLogin

```
procedure TFrLogin.btLoginClick(Sender: TObject);
var
  Qry: TSQLQuery;
begin
  Qry := TSQLQuery.Create(nil);
try
with Qry do
begin
      SQLConnection := dm.SQLConnection1;
      SQL.Text := ' SELECT ID, ADMINISTRADOR FROM USUARIO '+
            ' WHERE LOGIN = '+ QuotedStr(edLogin.Text) +
            ' AND SENHA = '+ QuotedStr(edSenha.Text);
      Open;
      UsuarioLogado := (not FieldByName('ID').IsNull);
      UsuarioID := FieldByName('ID').AsInteger;
      UsuarioAdministrador := (FieldByName('ADMINISTRADOR').AsString = 'S');
      UsuarioAtual := edLogin.Text;
      Close;
end;
  finally
    FreeAndNil(Qry);
end;
  Close;
end;
```

Foi declarada uma variável do tipo TSQLQuery (Qry) para a futura consulta na tabela USUARIO, assim como no método anterior, GetNovoCodigoAcesso(). Após criar a instância da classe para o objeto Qry, a propriedade SQLConnection do mesmo recebe o objeto responsável pela conexão com o banco de dados (SQLConnection1), presente no módulo dm, lembrando que o mesmo deve ser adicionado na seção Uses deste formulário para que possa ser referenciado. Conforme o comando SQL que foi definido na propriedade SQL.Text, se o login e a senha fornecidos pelo usuário através dos objetos do tipo TEdit forem encontrados, então o retorno do ID e do campo ADMINISTRADOR é efetuado. A variável definida anteriormente no módulo, UsuarioLogado, recebe o valor booleano através de um teste no campo ID, onde caso o valor do mesmo seja diferente de nulo (not[Campo].IsNull), o retorno é verdadeiro, e caso contrário, falso. Essa é a variável responsável por fornecer ao formulário principal a informação se o login foi corretamente efetuado ou não. Posteriormente as outras variáveis também recebem seus devidos valores através do retorno e dos dados informados. Para encerrar, o objeto Qry é encerrado, liberado da memória e o formulário de Login é fechado através do método Close().

Concluído formulário de Login, vamos a adição de quatro componentes do tipo TSpeedButton e um TMainMenu ao formulário principal. O objeto TMainMenu foi constituído pelo menu "Manutenção" e de um submenu "Cliente", possuindo também outro menu denominado "Usuário", que por sua vez está constituído pelo submenu "Acessos". Tanto os objetos do tipo TMenuItem, quanto os do tipo TSpeedButtons, com exceção do submenu denominado "Acessos", não tiveram códigos fontes definidos pois o foco deste projeto é apenas apresentar uma forma de controlar o acesso aos mesmos, considerando que pudessem ter qualquer funcionalidade. O formulário principal também recebeu um novo método que foi denominado BloquearComponentes(), cuja função é de desativar todos os componentes que são do tipo TMenuItem ou TSpeedButton em tempo de execução, contudo tal método pode ser facilmente alterado e empregado para bloquear quaisquer tipos de componentes de acordo com a necessidade do projeto em questão, lembrando que isto é possível através da propriedade Enabled de cada objeto. A **Listagem 4** apresenta o código do método BloquearComponentes().

Listagem 4. Código do método BloquearComponentes()

```
procedure TFrPrincipal.BloquearComponentes;
var
  i: Integer;
begin
for i := ComponentCount -1 downto 0 do
if Components[i] is TSpeedButton then
      TSpeedButton(Components[i]).Enabled := False
elseif Components[i] is TMenuItem then
      TMenuItem(Components[i]).Enabled := False;
end;
```

É efetuado um laçodo tipo "for",empregando a variável de controle "i", com o intuito de percorrer todos os componentes presentes no FrPrincipal, utilizando como base a propriedade ComponentCount, que por sua vez, armazena o número total de objetos que pertencem ao formulário. Em seguida, com o auxílio de Components, é possível testar se o objeto atual é do tipo TSpeedButton ou TMenuItem (ClassType), e caso o valor seja verdadeiro, tal objeto é desativado através de sua propriedade Enabled. Controles que podem armazenar outros controles são chamados de containers, eles disponilizam propriedades que expõem cada controle contido. No caso de um TForm, a propriedade é a Components, um array dos objetos existentes no formulário. O teste citado é feito através do operador "is", que realiza uma comparação de tipos.

Concluído o método responsável por bloquear todos os componentes, foi criado um novo procedimento com o intuito de liberar o acesso apenas aos componentes que o usuário logadono sistema possui. A **Listagem 5** apresenta o código do procedimento VerificarPermissoes(), cujo intuito é consultar a tabela USUARIOACESSSO e verificar quais são os objetos que o usuário tem acesso, utilizando também uma varredura em todos os componentes do formulário.

Inicialmente é efetuado um teste para verificar se o usuário atual é administrador, que por sua vez, tem acesso total e não são efetuadas as verificações assim como os bloqueios. Caso o usuário não seja administrador, então todos os componentes do tipo TSpeedButton e TMenuItem são bloqueados, através do método criado anteriormente, BloquearComponentes().

Em Seguida o sdsUsuario é posicionado no ID do usuário atual, para possibilitar a verificação dos acessos do mesmo utilizando a funçãoLocate(). Esta função muito útil presente em grande parte de componentes do tipo DataSet, permite a busca rápida de determinado valor em um determinado campo, podendo receber alguns parâmetros adicionais se assim for necessário e retorna um valor booleano, levando em consideração se a informação foi encontrada ou não. Posteriormente neste método é utilizado um laço do tipo "for" para efetuar a varredura em todos os componentes do formulário, e conforme a mesma é realizada, é efetuado um teste para verificar se a propriedade Name do componente atual (Componets[i].Name) está presente e se a permissão está ativa ("S"), e caso tal validação seja satisfatória, então é realizado um novo teste para verificar qual é a classe do componente atual e assim liberar o acesso ao mesmo para o usuário (Enabled := True), lembrando que para este exemplo toda vez que o nome de um objeto que pertence ao formulário está presente no campo COMPONENTE da tabela USUARIOACESSO e o valor do campo ATIVO é equivalente a "S", indica que tal componente é permitido ao usuário. Na função Locate() realizada no sdsUsuarioAcesso também é possível notar que a mesma permite que mais de um campo possa ser utilizado para efetuar a busca e consequentemente, a validação, onde neste exemplo foram utilizados os campos "ID" e "ATIVO" para que o retorno do mesmo fosse verdadeiro. Também é empregado o flagloCaseInsensitive, que por sua vez, não faz distinção entre letras maiúsculas e minúsculas para a pesquisa.

Listagem 5. Código do procedimento VerificarPermissoes() do FrPrincipal

```
FrPrincipalprocedure TFrPrincipal.VerificarPermissoes;
var
 i: Integer;
begin
  if not UsuarioAdministrador then
   begin
   BloquearComponentes;
   dm.sdsUsuario.Locate('ID', UsuarioID, []);
   for i := ComponentCount -1 downto 0 do
   begin
    if dm.sdsUsuarioAcesso.Locate('COMPONENTE; ATIVO',
      VarArrayOf([Components[i].Name,'S']),[loCaseInsensitive]) then
       if Components[i] isTSpeedButton then
        TSpeedButton(Components[i]).Enabled := True
       else if Components[i]is TMenuItem then
        TMenuItem(Components[i]).Enabled := True;
    end;
  end;
end;
```

Como o formulário principal foi mantido como o primeiro a ser carregado na aplicação, foi efetuada uma chamada ao formulário de Login no evento OnCreate do mesmo. A **Listagem 6** apresenta o código do evento OnCreate do FrPrincipal.

O formulário de Login é criado e exibido no momento da execução deste evento e em seguida, o mesmo é liberado da memória. Posteriormente, é verificada a variável UsuarioLogado presente no módulo e caso seu valor seja equivalente a falso, a aplicação deverá ser encerrada (Application.Terminate). É possível perceber também que a única maneira desta variável ter seu valor igual a verdadeiro é através da execução do código do evento OnClick do botão de login e evidentemente, da validação dos dados, implicando que se o mesmo não for executado, a aplicação será encerrada. Se o usuário está logado, então o procedimento VerificarPermissoes, definido anteriormente, é chamado.

Listagem 6. Código do evento OnCreate do formulário FrPrincipal

```
procedure TFrPrincipal.FormCreate(Sender: TObject);
begin
  FrLogin := TFrLogin.Create(nil);
  FrLogin.ShowModal;
  FreeAndNil(FrLogin);
if not UsuarioLogado then
Application.Terminate;
  VerificarPermissoes;
end;
```

Para dar continuidade a este projeto, foi adicionado outro formulário com o intuito de oferecer uma janela para que o usuário administrador possa definir quais são as permissões para cada usuário, sendo tal formulário denominado FrAcessos. Neste momento, para automatizar um pouco este processo manual, uma alternativa interessante que foi empregada é a criação, em tempo de execução, de objetos do tipo TCheckBox. O intuito deste processo é que a cada componente do tipo TSpeedButton ou TMenuItem presente no formulário principal, seja criado automaticamente um TCheckBox que represente o mesmo na tela de definição de acessos, tirando esta tarefa do desenvolvedor, facilitando também na futura adição de novos componentes dos tipos controlados. Neste exemplo, para que o TCheckBox criado represente determinado objeto presente no FrPrincipal, ele recebe o mesmo nome que foi definido para tal componente, uma vez que este nome também é utilizado para verificar as permissões de cada usuário através do campo COMPONENTE da tabela USUARIOACESSO. Antes de iniciar a definição do método que deve criar os componentes do tipo TCheckBox em tempo de execução, deve ser criado um outro procedimento para a manipulação do evento OnClick() dos mesmos. A **Listagem 7** apresenta o método responsável por manipular tal evento, que recebeu a denominação de GerenciarAcessos().

Como este método será empregado para o evento OnClick() de todos os objetos TCheckBox criados em tempo de execução, ele deve receber como parâmetro o objeto atual que está efetuando a sua chamada (Sender: TObject). No início deste procedimento, é verificado através de TCheckBox(Sender).Checked, se o objeto do tipoTCheckBox que invocou o método está com a propriedade Checked definida como verdadeiro, ou em outras palavras, se o mesmo está marcado. Se estiver, é efetuada uma busca para verificar se o mesmo já se encontra na tabela de acessos através do sdsUsuarioAcesso e da função Locate(), lembrando que o módulo dm deve ser incluído na seção Uses deste formulário. Se ele já está

cadastrado, então o sdsUsuarioAcesso recebe o comando Edit para permitir a edição do registro atual, que por sua vez recebe o valor de "S" no campo ATIVO, onde posteriormente a gravação é efetuada através do comando Post. Caso ele não esteja cadastrado, então o mesmo é inserido na tabela USUARIOACESSO através do sdsUsuarioAcesso e do método Append. O novo registro recebe um novo código, o valor ID do usuário atual, o valor do campo COMPONENTE equivalente ao nome do TCheckBox atual, que é o mesmo nome do componente presente no formulário principal, e o valor de "S" para o campo ATIVO, onde o comando Post é chamado para efetuar a gravação.

Listagem 7. Código do método GerenciarAcessos()

```
procedure TFrAcessos.GerenciarAcesso(Sender: TObject);
begin
 if TCheckBox(Sender).Checked then
 begin
  if dm.sdsUsuarioAcesso.Locate('COMPONENTE',TCheckBox(Sender).
   Name,[loCaseInsensitive]) then
  begin
   with dm do
   begin
    sdsUsuarioAcesso.Edit;
    sdsUsuarioAcessoATIVO.AsString := 'S';
    sdsUsuarioAcesso.Post;
   end;
  end
  else
  begin
   with dm do
   begin
    dm.sdsUsuarioAcesso.Append;
    sdsUsuarioAcessoID.AsInteger := GetNovoCodigoAcesso;
    sdsUsuarioAcessoIDUSUARIO.AsInteger := sdsUsuarioID.AsInteger;
    sdsUsuarioAcessoCOMPONENTE.AsString := TCheckBox(Sender).Name;
    sdsUsuarioAcessoATIVO.AsString := 'S';
    sdsUsuarioAcesso.Post;
   end;
  end;
 end
 else
  if dm.sdsUsuarioAcesso.Locate('COMPONENTE',TCheckBox(Sender).
   Name,[loCaseInsensitive]) then
 begin
  with dm do
  begin
   sdsUsuarioAcesso.Edit;
   sdsUsuarioAcessoATIVO.AsString := 'N';
   sdsUsuarioAcesso.Post;
  end;
 end;
end;
```

Caso o TCheckBox atual esteja desmarcado então é efetuada uma busca para encontrar o registro que corresponde ao mesmo, com o intuito de definir a propriedade ATIVO do mesmo com o valor de "N", indicando que ele não deve ter acesso a tal componente no formulário principal.

Com a conclusão do método GerenciarAcessos(), a próxima etapa é criar o procedimento que será responsável por criar os objetos do tipo TCheckBox de acordo com os componentes presentes no formulário principal. A **Listagem 8** apresenta o código do método responsável por esta tarefa, que foi denominado CriarComponentes().

Listagem 8.Método CriarComponentes()

```
procedure TFrAcessos.CriarComponentes(Formulario: TForm);
var
  CheckBox: TCheckBox;
  i, auxTop, auxLeft: Integer;
begin
  auxTop := 10;
  auxLeft := 20;
for i := Formulario.ComponentCount -1 downto 0 do
begin
   if (Formulario.Components[i] is TSpeedButton)
   or (Formulario.Components[i] is TMenuItem) then
begin
      Inc(auxTop, 20);
      CheckBox := TCheckBox.Create(FrAcessos);
      CheckBox.Parent := FrAcessos;
      CheckBox.Name := Formulario.Components[i].Name;
      CheckBox.Caption := Formulario.Components[i].ClassName +'-'+
                  Formulario.Components[i].Name;
      CheckBox.OnClick := GerenciarAcesso;
      CheckBox.Top := auxTop;
      CheckBox.Left := auxLeft;
      CheckBox.Width := 250;
   end;
  end;
end;
```

Este método recebe como parâmetro um formulário para que a busca pelos devidos componentes seja implementada, que futuramente será o FrPrincipal. Também são definidas algumas variáveis para auxiliar durante o processo de criação dos objetos em tempo de execução, sendo elas CheckBox, que deverá receber uma instância da classe TCheckBox, "i", variável de controle para o laço "for", auxTop e auxLeft, para auxiliar no posicionamento de cada TCheckBox que será criado através da definição dos valores das propriedades Top e Left do mesmo. Em seguida, as variáveis para o posicionamento são inicializadas com valores padrões. Posteriormente, um laço "for" é realizado com o intuito de percorrer os componentes do formulário em questão. Se o componente atual (Components[i]) é do tipo TSpeedButton ou TMenuItem, então a variável auxTop é incrementada com o valor vinte através do comando Inc(). É criada uma instância da classe TCheckBox sendo a mesma atribuída ao objeto CheckBox. Através da propriedade Parent, o CheckBox é definido como "filho" do formulário FrAcessos. A propriedade Name do TCheckBox atual recebe o mesmo nome do componente na qual ele representa, através de Formulario.Components[i].Name, lembrando que o nome é a propriedade que foi adotada neste exemplo para vincular o TCheckBox ao seu devido componente no formulário FrPrincipal e assim representá-lo na definição de acessos. Posteriormente na propriedade Caption é atribuído o nome da classe do componente atual do formulário e seu devido nome, apenas para facilitar na identificação do mesmo, uma vez que esta propriedade é exibida como descrição para o TCheckBox. No evento OnClick, é definido

o evento criado anteriormente, GerenciarAcesso(). Para concluir, o TCheckBox criado em tempo de execução é posicionado de acordo com as variáveis auxTop e auxLeft, tendo também sua largura definida (width).

Neste formulário de acesso também foram adicionados outros componentes, sendo eles dois TGroupBox e um TComboBox. Este último recebeu a denominação de cbUsuario e terá a finalidade de permitir ao administrador, selecionar qual o usuário que ele deseja definir os acessos. Para dar continuidade ao projeto, deve ser criado um novo método para possibilitar o carregamento das permissões de cada usuário no momento em que o formulário FrAcessos for carregado ou o comboboxcbUsuario for alterado. A **Listagem 9** apresenta o método responsável por esta funcionalidade, que foi denominado CarregarPermissoes().

Listagem 9. Método CarregarPermissoes()

```
procedure TFrAcessos.CarregarPermissoes;
var i: Integer;
begin
 for i := ComponentCount -1 downto 0 do
  if Components[i] is TCheckBox then
   if dm.sdsUsuarioAcesso.Locate('COMPONENTE;ATIVO',VarArrayOf
    ([Components[i].Name,'S']),[loCaseInsensitive]) then
     TCheckBox(Components[i]).Checked := True
   else
     TCheckBox(Components[i]).Checked := False;
end;
```

Após a criação dos objetos do tipo TCheckBox em tempo de execução, este método efetua uma varredura pelos componentes do formulário FrAcessos utilizando o laço "for" e a propriedade ComponentCount e verifica se o componente atual é do tipo mencionado (TCheckBox). Se ele for, então a função Locate() do objeto sdsUsuarioAcesso é chamada para verificar se um registro com o mesmo nome do componente atual existe para o campo COMPONENTE, testando também se ele está ativo (ATIVO = "S"), e de acordo com o resultado da pesquisa o objeto TCheckBox atual é marcado ou não, para que o administrador possa identificar quais são as permissões de acesso atuais para o usuário selecionado.

Com a conclusão do método CarregarAcessos(), o evento OnChange() do combobox cbUsuario deve ser definido para que o mesmo carregue quais são as permissões do usuário selecionado no momento em que o administrador alterar o valor do mesmo. Sendo assim, ele deve receber o trecho de código apresentado a seguir:

```
dm.sdsUsuario.Locate('LOGIN',cbUsuario.Text,[loCaseInsensitive]);
CarregarPermissoes;
```

É possível notar que primeiramente o sdsUsuario é devidamente posicionado para o usuário selecionado pelo administrador, através do campo LOGIN e o valor do combobox, e posteriormente o método responsável por carregar os acessos é chamado. Para concluir o formulário FrAcessos, resta apenas definir o código para o evento OnShow() do mesmo para carregar alguns valores no momento em que ele for exibido. A **Listagem 10** apresenta o código do evento OnShow() do formulário FrAcessos.

Listagem 10. Método CarregarPermissoes()

```
procedure TFrAcessos.FormShow(Sender: TObject);
begin
  dm.sdsUsuario.First;
while not dm.sdsUsuario.Eof do
  begin
    cbUsuario.Items.Add(dm.sdsUsuarioLOGIN.AsString);
    dm.sdsUsuario.Next;
end;
  cbUsuario.Text := UsuarioAtual;
  dm.sdsUsuario.Locate('LOGIN', cbUsuario.Text, [loCaseInsensitive]);
CarregarPermissoes;
end;
```

É efetuado um laço do tipo while no sdsUsuario com o intuito de percorrer todos os registros e adicionar o login de todos os usuários à lista do objeto TComboBoxcbUsuario, através do comando Items.Add(). Em seguida, o cbUsuario recebe o valor do usuário atual. Para concluir, o sdsUsuario também é posicionado no usuário atual e o método CarregarPermissoes() é chamado.

Um ponto importante para concluir o projeto é com relação à criação automática dos objetos que o compõe. Como o módulo foi utilizado para toda a conexão com o banco de dados, convém que o mesmo seja inicializado antes dos outros formulários do projeto para evitar erros, uma vez que o evento OnCreate() do formulário principal (FrPrincipal) foi alterado para chamar o formulário de Login, que por sua vez utiliza o objeto SQLConnection1. Tal processo pode ser efetuado facilmente através do menu Project>Options>Forms, que também permite que o formulário principal (inicial) da aplicação possa ser alterado. A **Figura 2** apresenta a definição da opção Forms deste projeto da janela Project Options.

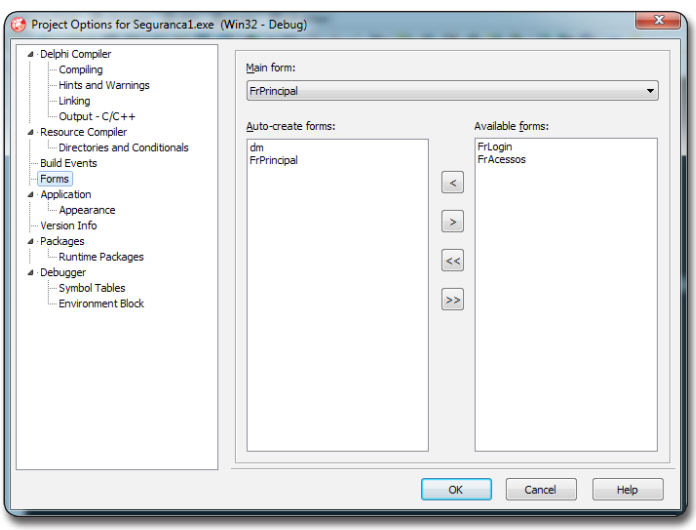

Figura 2. Configurações da janela Project Options

Neste momento, o projeto de implementação de Segurança através de controle de acesso está concluído. A **Figura 3** apresenta a aplicação em execução com a tela de acessos em evidência.

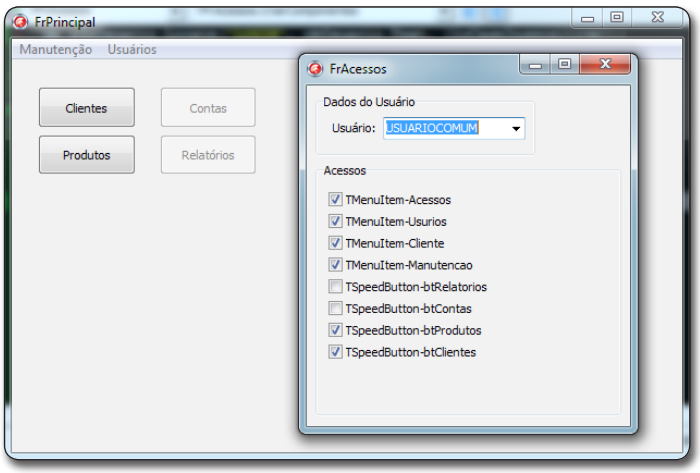

Figura 3. Aplicação em execução com tela de acessos em evidência

Como pode ser observado, desenvolver um controle de acessos de maneira manual é um processo que demanda certo tempo e trabalho. Também é importante frisar que este é um simples exemplo e uma ideia de como implementar o conceito de Segurança em aplicações, onde o mesmo deve ser devidamente adaptado e otimizado de acordo com a necessidade do projeto em questão.

Implementando segurança com o componente UserControl

Umaferramenta interessante para a implementação de segurança utilizando gerenciamento de usuários e controle de acessos para as aplicações de uma forma rápida e fácil, é o UserControl, que por sua vez é freeware e pode ser facilmente obtido (ver seção **Links**). Ele está disponível para diversas versões do Delphi, inclusive a XE 2. Como mencionado anteriormente, ele é composto por diversos componentes e recursos que permitem a integração com vários frameworks para servidores de banco de dados, onde as permissões ficam devidamente armazenadas em tabelas específicas no banco, criadas automaticamente pelo próprio componente. Ele também traz consigo, funcionalidades para efetuar Logs dos eventos que são realizados no sistema entre outros recursos, como é o caso do envio de emails para notificações.

Com relação aos componentes da aplicação que devem ser bloqueados e liberados aos usuários, o UserControl permite um controle de diversos tipos de objetos (menus, botões, etc) e as ações, onde as permissões podem ser definidas a níveis de usuários ou perfis. O componente principal para a utilização desta ferramenta nos projetos é o TUserControl. Através dele é possível definir as configurações de Login, os dados do usuário administrador, as configurações de Log, o vínculo com o componente de interligação para o banco de dados, dentre outras diversas configurações. Contudo, o UserControl também é composto por outros componentes importantes e necessários para a implementação dos recursos da ferramenta. A **Tabela 1** apresenta alguns dos principais componentes do UserControl além do TUserControl.

Contudo, para não estender, alguns componentes do User Controlempregados para a integração com outros frameworks de banco de dados não foram apresentados na **Tabela 1**, mas também estão disponíveis, como é o caso do TUCIBOConn, TUCFIBConn, TUCIBXConn, entre outros.

UserControl na prática

Considerando que o componente UserControl esteja devidamente instalado no Delphi, será criado um novo projeto. Assim como no primeiro exemplo, o formulário recebeu a denominação de FrPrincipal. Para esta segunda demonstração, será utilizado o mesmo banco de dados criado anteriormente, uma vez que o componente UserControl necessita de uma base de dados para armazenar as informações dos usuários e suas permissões. Também foram criados os mesmos objetos do exemplo anterior, um TMainMenu, constituído por quatro objetos do tipo TMenuItem, e quatro botões com o intuito de demonstrar como controlar os acessos aos mesmos, considerando que eles pudessem ter qualquer funcionalidade assim como no primeiro exemplo. Entretanto, foram adicionados dois novos submenus no menu Usuários, um denominado AlterarSenha e o outro, Logoff, para a futura configuração do componente UserControl1.

A próxima etapa deste segundo exemplo é inserir um componente do tipo TUserControl no formulário principal da aplicação, lembrando que este é o componente principal desta ferramenta e que centraliza o gerenciamento e configuração das permissões, onde apenas um objeto deste tipo é necessário por projeto para controlar todos os outros componentes. Vale à pena ressaltar que

Classe/Componente	Descrição
TUCSettings	Componente que permite o ajuste de diversas configurações e propriedades.
TUCControls	Um componente importante que permite ao desenvolvedor, selecionar quais os objetos do formulário (Botões, Edits, Labels)que os acessos serão controlados, possuindo uma janela para auxiliar neste processo.
TUCDBXConn	Componente para integração com o objeto encarregado da conexão com banco de dados do framework DBExpress, TSQLConnection.
TUCADOConn	Componente responsável pela interligação com banco de dados através de ferramentas do framework ADO.
TUCBDEConn	Componente para conexão através do framework BDE.
TUCZeosConn	Componente para conexão através do framework Zeos.
TMailUserControl	Componente para envio de email através do UserControl, permitindo uma monitoração dos usuários.

Tabela 1. Principais componentes do UserControl

ao inserir um objeto do tipo TUserControl, é possível acessar uma janela de configurações facilmente através de um clique duplo sobre o mesmo. Para dar continuidade, um componente do tipo TUCDBXConn também deve ser inserido no formulário principal para prover a interligação com a base de dados, uma vez que o banco de dados adotado nestes exemplos foi o Firebird com o framework DBExpress, lembrando que um objeto TSQLConnection também deve ser inserido e definido de forma idêntica a configuração efetuada no primeiro exemplo. Após a devida configuração do objeto SQLConnection1, a propriedade Connection do componente UCDBXConn1, inserido anteriormente, deve ser definida com o valor de tal objeto (SQLConnection1). Em seguida, para concluir a interligação com o banco de dados utilizando o UserControl, basta apenas alterar a propriedade DataConnector do objeto UserControl1, definindo a mesma com o valor de UCDBXConn1. Posteriormente, algumas propriedades do objeto UserControl1 também devem ser definidas para que o mesmo possa executar corretamente, sendo as mesmas definidas de acordo com o trecho a seguir:

```
ControlRight.MainMenu = MainMenu1
User.MenuItem = Acessos
UserPasswordChange.MenuItem = AlterarSenha
UserLogoff.MenuItem = Logoff
Login.InitialLogin.Password = senha
Login.InitialLogin.User = admin
AutoStart = True
```

A propriedade ControlRight.MainMenu define qual é o objeto do tipo TMainMenu que deve ser controlado pelo componente, a User.MenuItem é empregada para atribuir qual é o menu que deve ser usado para o acesso a janela de configurações dos usuário, a UserPasswordChange.MenuItem define qual é o menu para a troca de senha do usuário, a UserLogoff.MenuItem é utilizada para definir qual é o item de menu quer permite efetuar Logoff, em Login. InitialLogin devem ser definidos o usuário e a senha para o login inicial do administrador, e através de AutoStart, o UserControl é ativado no momento da execução da aplicação. Neste momento é possível notar que sem nenhum trecho de código, já é possível implementar alguns recursos na aplicação e que consequentemente, o tempo para desenvolvimento é economizado. Outro ponto é com relação às janelas prontas do UserControl, tais como a tela de login e de configuração dos usuários, que por sua vez são geradas automaticamente de acordo com as definições dos componentes da ferramenta. A **Figura 4** apresenta a aplicação em execução com as configurações efetuadas até o momento, demonstrando a janela de login criada automaticamente pelo componente UserControl.

Para dar sequência ao projeto, um componente do tipo TUC Controls deve ser adicionado ao formulário principal (FrPrincipal), lembrando que este componente é o responsável pelo gerenciamento de quais objetos serão controlados pelo UserControl, e que o mesmo pode ser inserido em cada formulário do projeto para a definição dos componentes monitorados que estão contidos nele. Para interligá-lo ao componente principal, sua propriedade UserControl deve ser configurada com o valor de UserControl1.

Para realizar a definição dos objetos que serão controlados, basta efetuar um clique duplo sobre o mesmo para que a janela de definições seja aberta. A **Figura 5** demonstra a janela para a atribuição dos objetos que terão suas permissões de acesso controladas para o formulário que contém o mesmo (FrPrincipal).

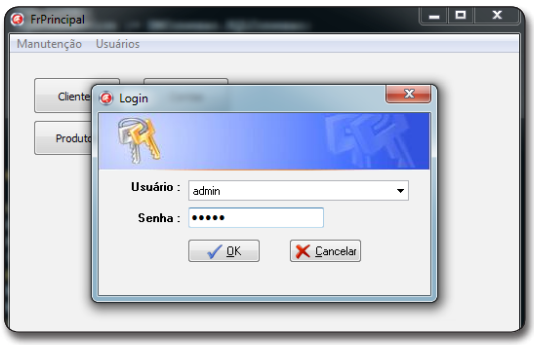

Figura 4. Aplicação em execução com janela de Login do UserControl

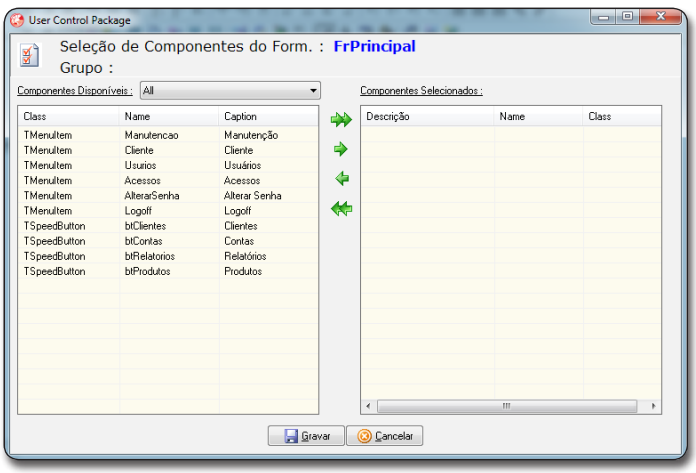

Figura 5. Janela para definição de componentes controlados pelo UserControl

Como pode ser observado na **Figura 5**, esta janela permite que o desenvolvedor informe de uma maneira rápida e prática quais são os objetosque deseja oferecer ao usuário a possibilidade de bloquear ou liberar, simplesmente selecionando o mesmo em "Componentes Disponíveis" e clicando no botão "Selecionar", ou ainda "Selecionar Todos". Através desta janela, o desenvolvedor também pode criar uma descrição para a tarefa executada pelo componente selecionado, facilitando para o usuário final.

Neste momento, ao executar a aplicação, é possível notar que o UserControl cria automaticamente uma janela para a configuração dos acessos para os usuários. A **Figura 6** apresenta a janela de configuração de permissões do UserControl em tempo de execução.

Como é possível observar neste exemplo, o UserControl permite aumentar consideravelmente a produtividade quando o assunto é implementação de segurança, uma vez que traz diversos recursos prontos para esta tarefa e sem nenhum trecho de código adicional.

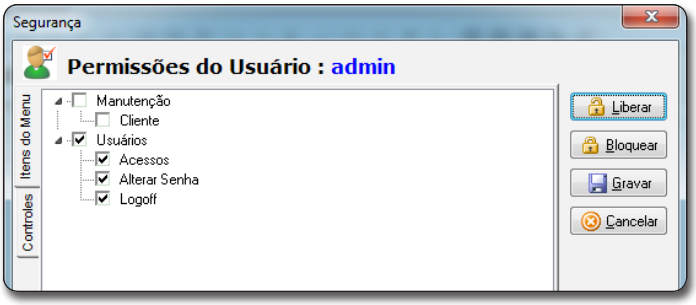

Figura 6. Janela para definição de permissões para o usuário selecionado

O pacote de componentes TMS Security System

O TMS Security System também é um pacote constituído por diversos componentes para que o processo de implementação de segurança nas aplicações seja automatizado. Desenvolvido pela TMS Software, este componente é proprietário e sua licença tem um custo, porém, uma versão Trial pode ser obtida facilmente através do site da empresa para que o desenvolvedor possa conhecer o mesmo (ver seção **Links**). Através deste conjunto de ferramentas, o desenvolvedor consegue definir de uma forma rápida, quais são os objetos que devem ter suas permissões controladas através dos editores presentes nos mesmos. A **Tabela 2** apresenta os componentes que pertencem ao TMS Security System.

Outro ponto importante com relação aos componentes do TMS Security System é quanto à possibilidade de personalização das janelas criadas automaticamente pelo mesmo, onde é possível definir os textos para os botões e labels que são apresentados.

Quanto ao funcionamento, os componentes do TMS Security System também armazenam as informações dos usuários em tabelas no banco de dados, que por sua vez podem ser criadas pelo desenvolvedor, onde a estrutura de campos das mesmas deve ser definida de acordo com o manual que acompanha a ferramenta, lembrando que posteriormente cada campo deve ser configurado em cada componente que necessita desta atribuição para a interligação com sua devida tabela, como é o caso do TuilSecurity Manager. Tal interligação é realizada através de componentes do tipo TDataSource, permitindo assim que diversos frameworks para a conexão possam ser utilizados.

TMS Security System na prática

Para implementar os recursos do conjunto de componentes do TMS Security System neste terceiro exemplo, a mesma base de dados utilizada anteriormente foi empregada. Como as tabelas devem ser devidamente criadas manualmente pelo desenvolvedor com auxílio do manual, a **Listagem 11** apresenta um código SQL que pode ser empregado para definí-las para o TMS Security System.

Listagem 11. Código SQL para criação das tabelas do TMS Security System

```
CREATE DOMAIN BOOLEAN AS SMALLINT CHECK (VALUE IS NULL OR VALUE IN(0, 1));
CREATE TABLE USUARIOS (
  ULOGINNAME VARCHAR(50) NOT NULL PRIMARY KEY,
  UFULLNAME VARCHAR(50),
  UPASSWORD VARCHAR(50),
  ULASTACCESS TIMESTAMP,
  UENABLED BOOLEAN,
  UCREATEDDATE TIMESTAMP,
  ULASTACCTIME TIMESTAMP,
  UCREATEDTIME TIMESTAMP,
  UACCESSCOUNT INTEGER
);
CREATE TABLE ACESSOS_USUARIO (
  AUUSERNAME VARCHAR(50),
  AUPERMISSION VARCHAR(50),
  PRIMARY KEY (AUUSERNAME, AUPERMISSION)
);
CREATE TABLE GRUPOS (
  GGROUPNAME VARCHAR(50) NOT NULL PRIMARY KEY,
  GDESCRIPTION VARCHAR(50)
);
CREATE TABLE ACESSOS_GRUPO (
  AGGROUPNAME VARCHAR(50),
  AGPERMISSION VARCHAR(50),
  PRIMARY KEY(AGGROUPNAME, AGPERMISSION)
);
CREATE TABLE MEMBROS_GRUPO (
  MGUSERNAME VARCHAR(50),
  MGGROUPNAME VARCHAR(50),
  PRIMARY KEY(MGUSERNAME, MGGROUPNAME)
);
CREATE TABLE PERMISSOES (
  PPERMISSION VARCHAR(50) NOT NULL PRIMARY KEY,
  PITEMS BLOB SUB_TYPE TEXT SEGMENT SIZE 80,
  PACTION INTEGER,
  PFORMNAME VARCHAR(50)
);
```

Classe/Componente	Descrição
TuilSecurityManager	Componente principal do TMS Security System, responsável por manter as informações de usuários e suas permissões, devendo os outros componentes se interligarem ao mesmo. Apenas um componente deste tipo é necessário por projeto.
TuilFormPolicy	Componente responsável pela definição da segurança do formulário, permitindo estabelecer os componentes e ações que serão controlados, possuindo um editor para facilitar o processo.
TuilLoginDlg	Componente responsável pela janela de login para a autenticação dos usuários, onde o método Execute() do mesmo deve ser chamado na inicialização da aplicação.
TuilChangePasswordDlg	Componente destinado a janela de alteração de senha do usuário, devendo ser chamado também através do método Execute().
TuilFormPolicyDlg	Componente para a definição das políticas de acesso. Também é chamado através do método Execute().
TuilSecurityDlg	Componente responsável pela janela de gerenciamento dos usuários.

Tabela 2. Principais componentes do TMS Security System

Como pode ser observado neste trecho de código, foi criado um default com a denominação de Boolean para simular um campo booleano e posteriormente, foram criadas seis novas tabelas com os campos definidos de acordo com as informações do manual, onde as chaves primárias foram implementadas diretamente em campos do tipo VARCHAR apenas para facilitar no desenvolvimento do exemplo.

Concluída a criação do banco de dados, a próxima etapa é iníciar um novo projeto VCL. Como o banco de dados adotado foi o Firebird, assim como nos outros exemplos, um objeto do tipo TSQLConnection deve ser inserido e configurado conforme apresentado anteriormente. Em seguida, para cada tabela, também devem ser criados objetos do tipo TSimpleDataSet, sendo ambos configurados de forma semelhante a que foi apresentada, ressaltando que o comando ApplyUpdates() deve ser utilizado nos eventos AfterDelete() e AfterPost() dos mesmos para efetivar as gravações que forem efetuadas. Também devem ser inseridos objetos do tipo TDataSource para cada TSimpleDataSet. Terminada a inserção dos seis objetos TSimplesDataSet e suas definições, também foram inseridos quatro TSpeedButtons e um TMainMenu de forma idêntica aos outros exemplos, apenas para demonstrar como controlar os acessos. Para dar início a implementação de segurança com o TMS Security System, um objeto do tipo TuilSecurityManager deve ser inserido no formulário. Após sua inserção, as subpropriedades GroupBindary, PermissionBindary e UserBindary do mesmo precisão ser definidas. Em GroupBindary, são estabelecidos todos os DataSets (através da subpropriedade DataSource) pertinentes aos grupos de usuários (Tabelas GRUPOS, ACESSOS_GRUPO e MEMBROS_GRUPO), bem como todos os campos, onde o desenvolvedor informa qual subpropriedade do componente está relacionada ao campo da tabela, como por exemplo em GroupMembership o valor será equivalente a GDESCRIPTION, e assim sucessivamente. Em PermissionBindary é definido o DataSet referente a tabela que armazena os componentes que serão controlados pelo TMS Security System (PERMISSOES), bem como as subpropriedades referentes a cada campo. Para finalizar a configuração, em UserBindary são definidos os DataSets relacionados as tabelas dos usuários (USUARIOS e ACESSOS_USUARIO), bem como as subproprieades e seus campos. Como é possível notar, este componente é utilizado para centralizar todas as conexões com os TDataSets através de seus respectivos TDataSources, possuindo também um editor para cadastrar facilmente os usuários e os grupos através de um clique duplo sobre o mesmo.

Para dar sequência ao exemplo, um componente do tipo TuilLogin Dlg deve ser inserido no formulário principal para possibilitar a autenticação do usuário no momento da execução da aplicação. Sua propriedade SecurityManager deve ser configurada com o valor de SecurityManager1. Dentre as outras propriedades, além das utilizadas para alterar o Caption dos botões e os títulos, duas que merecem destaque são a UseWindowsUserName e Retries. Em UseWindowsUserName, o desenvolvedor define se o TMS Security System deve obter automáticamente a informação de Login do Usuário atual no Windows, e em Retries, o número de vezes que o usuário tem para efetuar login, caso o mesmo erre a senha por exemplo. Após a definição destas propriedades, o evento OnCreate() do formulário principal deve ser alterado para possibilitar o correto carregamento do componente TuilLoginDlg. A **Listagem 12** apresenta o código utilizado no evento OnCreate() do FrPrincipal.

Listagem 12. Código do evento OnCreate() do FrPrincipal

```
procedure TFrPrincipal.FormCreate(Sender: TObject);
begin
  SQLConnection1.Open;
  sdsUsuarios.Open;
  sdsAcessosUsuario.Open;
  sdsPermissoes.Open;
  sdsGrupos.Open;
  sdsMembrosGrupo.Open;
  sdsAcessosGrupo.Open;
  if not uilLoginDlg1.Execute then
    Application.Terminate;
end;
```

Neste trecho de código é possível notar que a conexão e os datasets são iniciados para a ligação com as tabelas do banco de dados e, posteriormente, o método Execute() do objeto uilLoginDlg1 é chamado para a exibição e autenticação do usuário. Caso o usuário não seja devidamene autenticado, a aplicação é encerrada através de Application.Terminate().

O próximo componente que deve ser inserido no formulário principal é o TuilSecurityDlg. Como mencionado anteriormente, este componente é responsável por exibir a janela para definição das permissões dos usuários. Ele também possui a propriedade SecuritySystem, que por sua vez deve ser configurada com o valor de SecuritySystem1. Também é possível alterar o título da janela e de diversos componentes que a compõe, onde por padrão os valores estão definidos em inglês. Terminada a configuração deste componente, ele pode ser facilmente chamado para exibir a janela de definições com a utilização do evento OnClick() de um menu ou outro objeto, por exemplo, através do método a seguir:

```
uilSecurityDlg1.Execute;
```

De fato, a implementação deste componente é extremamente rápida e fácil. Para dar continuidade a este exemplo, dois novos componentes devem ser inseridos, o TuilFormPolicy e o TuilFormPolicyDlg. O TuilFormPolicy é o componente responsável por definir os objetos (botões, menus, etc) que deverão ter seus acessos controlados, onde tais objetos podem ser adicionados por um editor através de um clique duplo sobre o mesmo em tempo de design, devendo o objeto TuilFormPolicy estar interligado ao SecuritySystem1 através da sua propriedade SecuritySystem. Um detalhe importante sobre este componente é que o desenvolvedor deve adicinar um nova política através do botão "Add Policy" do editor, atribuir um nome para a mesma e assim adicionar um ou mais componentes que fazem parte de tal política para que o

usuário administrador, em tempo de execução, defina os devidos acessos aos usuários. Já o componente TuilFormPolicyDlg permite a exibição da janela de atribuição dos objetos controlados em tempo de execução, devendo a propriedade FormPolicy do mesmo ser definida com o valor de uilFormPolicy1. Sua execução também pode ser efetuada de uma forma simples através do método Execute().

Neste momento, este simples exemplo está concluído. A **Figura 7** apresenta a aplicação em execução com dois usuários cadastrados e uma política de acesso que recebeu a denominação de "Botoes".

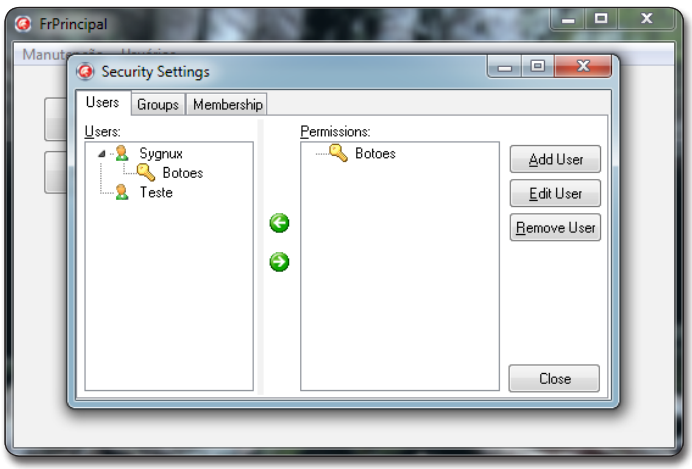

Figura 7. Aplicação baseada no TMS Security System em execução

Como é possível notar neste exemplo, o TMS Security System, assim como o UserControl, é uma ferramenta que permite automatizar e agilizar o processo de implementação de segurança nas aplicações, uma vez que possuem diversos recursos prontos para serem empregados.

Conclusão

A segurança, no ambiente computacional, é um fator que depende de diversos detalhes, onde uma análise detalhada do sistema como um todo deve ser realizada. É evidente que o executável da aplicação deve estar protegido e restrito ao máximo, mas os outros recursos envolvidos também não podem ser desconsiderados quando o assunto é aumentar os níveis de integridade dos dados. O emprego de criptografia para as informações que são armazenadas na base de dados é um exemplo importante de segurança adicional. Afinal, se por algum motivo alguém conseguir acessar tais informações diretamente, se depara com vários caracteres que parecem não fazer o menor sentido e, dificilmente, encontra uma forma de decifrá-los.

Para empregar segurança nas aplicações, o principal método adotado é a autenticação dos usuários através de uma janela de login durante a inicialização, verificando quais são as permissões de acesso dos mesmos de acordo com as políticas que foram definidas por um administrador. Como foi possível observar, este processo pode ser efetuado de uma maneira manual, onde tabelas, formulários e componentes devem ser criados, bem como deve ser definido todo o código fonte necessário para efetuar tal gerenciamento, implicando em uma demanda maior de tempo. Entretanto, existem diversos componentes importantes que automatizam e simplificam este processo, gratuitos e pagos, tais como o UserControl e o TMS Security System. Com suas telas e funções prontas, tais ferramentas auxiliam ao máximo a implementação e a futura manutenção das aplicações, uma vez que novos formulários e componentes podem ser adicionados ao projeto e também deverão ter seus acessos definidos.Sendo assim, normalmente a melhor alternativa para empregar segurança nas aplicações é utilizar ferramentas de terceiros, pois poupam tempo e esforços, mas evidentemente tudo vai depender da necessidade do projeto em questão.

Giuliano Scombatti Pinto
giuliano@sygnux.com.br
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Informações e Download do UserControl
http://code.google.com/p/usercontrolpackage/

Site da TMS Software com informações sobre o TMS System Security e download
http://www.tmssoftware.com/site/tss.asp

DevSpace do Autor
http://www.devmedia.com.br/space.asp?id=283274

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

FastReport – Parte 1

Elaboração de Relatórios Passo a Passo

Este artigo faz parte de um curso

Resumo DevMan

De que se trata o artigo:

Este artigo trata de demonstrar como utilizar o gerador de relatórios FastReport, tanto na versão integrada ao ambiente de desenvolvimento do Delphi XE 2 como a completa, com a finalidade de exibir informações da base de dados aos usuários de uma forma rápida e prática, apresentando o editor visual e seus principais componentes.

Em que situação o tema é útil:

O tema é útil em praticamente qualquer aplicação, já que é uma prática muito comum criar relatórios para ilustrar informações por meio de texto comum ou mesmo graficamente, permitindo assim uma maior interatividade e análise entre o usuário e a aplicação.

FastReport :

Imaginar um Software de uma empresa que não apresenta um acesso rápido e fácil às informações, ou em outras palavras, a exibição de relatórios detalhados, é praticamente inconcebível. Os relatórios são de extrema importância para qualquer seguimento do mercado, desde o mais simples até o mais sofisticado. Através dos mesmos é possível analisar e tomar as decisões corretas para o futuro da empresa. Diante deste fator, existem várias ferramentas disponíveis, especializadas na elaboração de relatórios no Delphi e outros ambientes de desenvolvimento também, como é o caso da ferramenta tema deste artigo. O FastReport é uma ferramenta consolidada que está presente há muito tempo no mercado e possui uma versão que faz parte do conjunto de recursos do Delphi XE 2, podendo ser facilmente empregado no desenvolvimento de relatórios para as mais variadas aplicações. Neste artigo serão apresentadas as características principais desta ferramenta, bem como seus componentes e integrações necessárias para que o mesmo possa ser utilizado na apresentação de um simples relatório de contas à receber, detalhado em três níveis, com o emprego de recursos que o FastReport oferece.

A informação, quando coerente, é o pilar principal para que quaisquer aplicações possam alcançar e demonstrar os resultados esperados pelos usuários finais, auxiliando os empresários a tomar decisões importantes no cotidiano que são vitais para a manutenção e prosperidade da organização de uma forma geral. Evidentemente, além da base de dados, a apresentação de relatórios é uma obrigatoriedade para que qualquer Software destinado ao ambiente comercial ou industrial possa obter uma grande aceitação pelos seus utilizadores, relatórios estes que podem ir desde os mais simples até os estratégicos, que por sua vez podem dar uma visão mais abrangente da situação da empresa através de comparações e estatísticas por exemplo. Além disso, os relatórios permitem que as informações sejam expostas de uma forma rápida, organizada e de fácil interpretação. O Delphi XE 2 conta com diversas ferramentas para a elaboração de relatórios, onde muitas são pagas e outras gratuitas, possuindo também algumas que podem ser instaladas junto com o mesmo, como é o caso do FastReport 4 Embarcadero Edition. O FastReport é uma ferramenta profissional com várias funcionalidades que fazem parte do mercado de elaboração de relatórios há mais de 10 anos, possuindo também outras versões mais completas com outros recursos do que a presente no Delphi XE 2, como é o caso da Enterprise. Outro fator interessante é que o mesmo pode ser encontrado para outras plataformas, como .NET, e versões anteriores do próprio Delphi, lembrando que as mesmas não são gratuitas. Quanto ao desenvolvimento, o FastReport possui um editor visual próprio customizável que permite a elaboração de relatórios de uma forma rápida e prática. Por padrão, tais relatórios são gravados em conjunto com os próprios formulários, nos arquivos do tipo DFM (ver **nota do DevMan 1**), mas também podem ser salvos em arquivos diretamente em um meio físico como o HD, se assim for o desejo do desenvolvedor, que posteriormente, pode integrar rapidamente sua aplicação ao arquivo referente ao relatório e assim permitir a pré-visualização, exportação e impressão do mesmo facilmente, através dos vários componentes da ferramenta. O Editor também permite que um relatório possa ser pré-visualizado em tempo de Design.

O FastReport oferece suporte à diversos objetos, tais como imagens, textos, gráficos, códigos de barra, checkboxes, linhas, tabelas, formas, entre outros, bem como permite a programação de códigos internos nas linguagens PascalScript, JScript, BasicScript e C++Script. Outra característica relevante é com

relação aos componentes capacitados a exportar o relatório para diversos formatos de arquivos, como é o caso de imagens (JPEG, GIF e BMP), documentos (HTML, XML, PDF, DOC, XLS, ODT, ODS, etc), entre outros, o que torna o mesmo fácil de ser reutilizado ou compartilhado pelos seus utilizadores. Ao empregá-lo nas aplicações, o desenvolvedor pode enviar parâmetros em tempo de execução para o relatório antes de sua exibição, onde é possível informar valores de variáveis, adicionar componentes diretamente através de código fonte e etc. Também possui funcionalidades destinadas a depuração (breakpoints).

Quanto aos recursos pertinentes às ferramentas de relatório, o FastReport também possui bandas (DataBands) com vários níveis e agrupamentos, componentes para cálculo de funções de agrupamento como SUM, suporte a variáveis, Querys internas (na versão completa), dentre outros, possuindo também uma característica interessante que é a herança para relatórios, onde o desenvolvedor pode herdar vários dados em comum para todos os relatórios, como por exemplo, um cabeçalho com um logotipo e os dados da empresa, uma informação que normalmente faz parte de todos os relatórios presentes no Software.

Dentre outras vantagens, o FastReport é uma ferramenta compacta e de baixo consumo de recursos, podendo ser utilizado em máquinas antigas e com menor Hardware. Também suporta a concepção de relatórios direcionados às impressoras matriciais, permitindo que os mesmos possam ser impressos corretamente e de forma rápida.

Além dos vários recursos apresentados, versões mais recentes do FastReport também contam com um conversor para relatórios que foram desenvolvidos no Rave Reports, outra ferramenta que acompanha o Delphi XE 2 para a elaboração de relatórios, evitando assim o retrabalho por parte do desenvolvedor que deseja migrar para tal ferramenta.

Com relação as suas novidades, recentemente foi lançada uma versão para o FireMonkey, que foi denominada FastReport FMX, sendo a mesma multiplataforma, permitindo a geração de relatórios para Windows e Mac OS X até o momento. Uma versão de avaliação pode ser obtida facilmente através do site do fabricante (ver seção **Links**).

Seu conjunto completo de ferramentas também inclui outras aplicações, como é o caso do FastScript e do FastQueryBuilder, que por sua vez não estão presentes na versão que acompanha o Delphi XE 2. O FastScript tem o objetivo de adicionar funcionalidades de scripts nas aplicações, sendo uma ferramenta multiplataforma, disponível para diversas linguagens de programação, assim como o Delphi. Já o FastQueryBuilder, como seu próprio nome sugere, é uma ferramenta voltada para a criação de consultas ou comandos SQL através de um construtor visual com o intuito de simplificar e agilizar este processo.

De uma forma geral, é um dos melhores geradores de relatório da atualidade, com várias funcionalidades como mencionado, pioneiro em diversos recursos e constantemente atualizado, razões que justificam sua aprendizagem e adoção nos projetos.

Componentes do FastReport

No Delphi XE 2, os componentes do FastReport podem ser rapidamente acessados através da paleta de ferramentas. O componente principal para a utilização da ferramenta é o TfrxReport. Ele é o responsável por permitir ao desenvolvedor abrir o editor visual em tempo de Design através de um clique duplo, e assim iniciar a criação do relatório, sua gravação, carregamento e alteração, bem como a pré-visualização e a impressão do mesmo em tempo de execução. Sendo assim, este componente possui diversos métodos importantes para a utilização dos relatórios. A **Tabela 1** apresenta alguns dos métodos mais utilizados do componente TfrxReport.

Nota do DevMan 1

No Delphi, um arquivo do tipo DFM tem a função de armazenar as informações sobre os componentes que estão presentes no formulário, salvando informações pertinentes a suas configurações como é o caso das propriedades. Este arquivo é atualizado automaticamente durante o processo de desenvolvimento toda vez que um objeto é adicionado ou tem suas características alteradas. Seu conteúdo pode ser facilmente visualizado ao clicar com o botão direito do mouse sobre o formulário e selecionar a opção "View as Text".

Método	Descrição
Clear	Utilizado para limpar um relatório.
LoadFromFile	Permite que um arquivo de relatório localizado em um meio físico como o HD por exemplo, possa ser carregado e futuramente impresso, onde o endereço do mesmo deve ser informado como parâmetro.
LoadFromStream	Oferece a opção de carregar um relatório através de uma Stream.
SaveToStream	Permite a gravação do relatório em uma Stream (ver **nota do Devman 2**).
PrepareReport	Permite a inicialização de um relatório sem a janela de pré-visualização.
ShowPreparedReport	Exibe um relatório, podendo ser utilizado após a chamada ao método PrepareReport.
ShowReport	Tem a função de iniciar e exibir um relatório em uma janela de pré-visualização, permitindo que o mesmo possa ser impresso ou exportado se o recurso estiver disponível.
Print	Efetua a impressão de um relatório.
Export	Permite que o relatório carregado possa ser exportado facilmente para um ou mais dos diversos tipos de arquivos suportados pelo FastReport, como é o caso do PDF.

Tabela 1. Alguns dos principais métodos do componente TfrxReport

Nota do DevMan 2

Um Stream pode ser definido como um fluxo de dados que está presente na memória, sendo constituído por bytes que possuem um início e um fim, podendo ser empregado para armazenar e manipular rapidamente as mais diversas informações, como textos, imagens, vídeos, entre outros.

Nota do DevMan 3

O TeeChart é um componente de terceiros que está integrado ao conjunto de ferramentas do Delphi com a finalidade de prover o desenvolvimento de diversos tipos de gráficos de uma forma fácil e rápida para as aplicações através de seu editor, onde as fontes de dados podem ser DataSets, Arrays, entre outros.

Contudo, o FastReport é uma ferramenta que traz consigo diversos recursos úteis que também podem ser empregados na aplicação, onde a maior parte deve ser utilizada em conjunto com o TfrxReport, adicionando assim funcionalidades, como permitir a integração com os DataSets do formulário e exportação do relatório. A **Tabela 2** demonstra alguns dos principais componentes do FastReport.

O FastReport também conta com alguns componentes adicionais para serem empregados em um ambiente de rede baseado no protocolo HTTP, em suas versões mais completas. O TfrxReportServer permite a criação de um servidor de relatórios e HTTP, não necessitando de componentes adicionais de terceiros para seu funcionamento. Já o TfrxServerConnection é o responsável por permitir a conexão com um servidor de relatórios (TfrxReportServer) com um ou mais clientes, sendo que o componente responsável pela conexão cliente é o TfrxReportClient, que efetua a comunicação com o mesmo e pode receber um relatório preparado direto do servidor, possibilitando ser impresso através da aplicação cliente. Este conjunto de ferramentas também conta com outro componente, o TfrxHTTPClient, que pode receber qualquer arquivo com base no protocolo HTTP.

Tutorial

FastReport na Prática

Para dar início a utilização do FastReport, um novo projeto VCL será criado (File>New>VCL Forms Application). Também serão criadas três simples tabelas (PESSOA, CONTASRECEBER e PARCELAS) utilizando o banco de dados Firebird, com o intuito de fornecer os dados de base para a geração do relatório. A **Listagem 1** apresenta o código SQL empregado na criação das tabelas. O banco de dados você pode salvá-lo em um local de fácil acesso e com o nome que achar melhor.

Concluída a criação das tabelas, a próxima etapa é inserir um objeto TSQLConnection para prover a conexão com o banco de dados utilizando o framework dbExpress, definindo a propriedade Driver do mesmo como "Firebird", LoginPrompt como "False", Params.Database apontando para o endereço da base de dados e então "Connected" como True.

Posteriormente, serão inseridos três objetos do tipo TSimpleDataSet para a conexão com as tabelas criadas anteriormente, que por sua vez, tiveram suas propriedades Connection definidas com o SQLConnection1 (o objeto de conexão), Name para seus devidos nomes (sdsPessoa, sdsContasReceber e sdsParcelas) e DataSet.CommandText para o código SQL relativo a sua devida tabela, como por exemplo "SELECT * FROM PESSOA". Vale à pena lembrar que os campos das tabelas devem ser adicionados aos DataSets através de um clique duplo sobre os mesmos e do comando "Add all fields", presente no menu de contexto das janelas de cada DataSet. Essa operação adiciona nos DataSets objetos que representam cada campo retornado pela instrução SELECT, permitindo também o recurso de autocomplete no editor de código, quando formos acessar algum campo.

Também foram adicionados três componentes TDataSource para a interligação entre os DataSets e receberam respectivamente o nome de dsPessoa, dsContasReceber e dsParcelas, onde a propriedade

Componente/Classe	Descrição
TfrxPreview	Permite ao desenvolvedor criar uma janela de pré-visualização customizada.
TfrxDesigner	Oferece ao usuário a funcionalidade de personalizar o relatório, podendo alterar a aparência visual do mesmo ou editá-lo.
TfrxDBDataSet	É um componente de conexão que permite que os DataSets ou DataSources presentes no projeto em Delphi possam ser interligados ao relatório, suportando diversos componentes como DBX, ADO, BDE, entre outros.
TfrxUserDataSet	Destinado a utilização de outros tipos de dados, como é o caso de arquivos, arrays, entre outros.
TfrxChartObject	Permite que um gráfico seja incorporado a um relatório, sendo baseado na biblioteca TeeChart (ver **Nota do DevMan 3**) presente no Delphi.
TfrxPDFExport	Utilizado para exportar um relatório para o formato PDF.
TfrxXLSExport	Permite a exportação de um relatório para um arquivo do Excel.

Tabela 2. Alguns dos principais componentes do FastReport

DataSet dos mesmos foi definida para cada TSimpleDataSet criado anteriormente. O próximo passo é definir o relacionamento entre ambos os DatasSets, onde o objetivo é criar um relatório de três níveis, contendo Master (sdsPessoa), Detail (sdsContasReceber) e SubDetail (sdsParcelas).

Listagem 1. Código SQL para as tabelas do exemplo

```
CREATE TABLE PESSOA (
  ID INTEGER NOT NULL,
  NOME VARCHAR(50),
  ENDERECO VARCHAR(50),
  CONSTRAINT PKY_PESSOA PRIMARY KEY(ID)
);
CREATE TABLE CONTASRECEBER (
  ID INTEGER NOT NULL,
  IDPESSOA INTEGER NOT NULL,
  DATAEMISSAO TIMESTAMP,
  DATAVENCIMENTO TIMESTAMP,
  FORMAPGTO VARCHAR(5) DEFAULT 'VISTA',
  NROPARCELAS INTEGER,
  VALORTOTAL NUMERIC(9,2),
  STATUS VARCHAR(1),
  CONSTRAINT PKY_CONTASRECEBER PRIMARY KEY(ID),
  CONSTRAINT FKY_CR_PESSOA FOREIGN KEY(IDPESSOA) REFERENCES
   PESSOA(ID)
);
CREATE TABLE PARCELAS (
  IDCONTASRECEBER INTEGER NOT NULL,
  NROPARCELA INTEGER NOT NULL,
  DATAEMISSAO TIMESTAMP,
  DATAVENCIMENTO TIMESTAMP,
  VALORPARCELA NUMERIC(9,2),
  STATUS VARCHAR(1),
  CONSTRAINT PKY_PARCELAS PRIMARY KEY(IDCONTASRECEBER, NROPARCELA),
  CONSTRAINT FKY_PA_CONTASRECEBER FOREIGN KEY(IDCONTASRECEBER)
   REFERENCES CONTASRECEBER(ID)
);
```

Sendo assim, na propriedade MasterSource do objeto sdsContasReceber, foi definido o valor de dsPessoa, em MasterFields, o valor "ID", e em IndexFieldNames "IDPESSOA", criando assim o vínculo da conta com a tabela pessoa. Também deve ser efetuada a ligação entre as parcelas e sua devida conta, onde a propriedade MasterSource do objeto sdsParcelas recebe o valor de dsContasReceber, MasterFields o valor "ID" e IndexField Names, "IDCONTASRECEBER". Vale à pena ressaltar também que as tabelas devem ser alimentadas através de inserções no próprio banco ou conforme critério do desenvolvedor.

O próximo passo para o projeto é adicionar um componente TfrxReport ao formulário, para dar início a criação do relatório e posteriormente, seu carregamento. Também devem ser adicionados três objetos do tipo TfrxDBDataSet no formulário com o intuito de efetuar a ligação do relatório com os três DataSets do dbExpress que foram adicionados anteriormente, onde eles receberam as denominações de frxdbPessoa, frxdbContasReceber e frxdbParcelas, através das propriedades Name e UserName. Para concluir a configuração dos mesmos, basta configurar a propriedade DataSource de cada um para seu DataSource equivalente.

Com os objetos básicos para este exemplo devidamente inseridos, basta dar um clique duplo sobre o frxReport1 para a abertura do editor do FastReport. Após o carregamento do mesmo, uma página em branco é exibida e o relatório já pode ser elaborado. Neste exemplo, foi adicionada uma banda do tipo Report Title para abrigar o título do relatório, que pode ser inserida facilmente através do botão Insert Band , presente na barra de ferramentas. Em seguida foi adicionado um simples texto ao cabeçalho, que pode ser inserido através do botão TextObject, também presente na barra de ferramentas do editor.

Criado o título do relatório, é hora de integrar os DataSets ao mesmo. Para acessar os DataSets do formulário, basta utilizar a opção Data, presente no menu Report do editor. A **Figura 1** apresenta a janela "Select Report DataSets" que é aberta contendo a lista de DataSets disponíveis ao efetuar este processo.

Sendo assim, basta selecionar todos os DataSets apresentados e clicar em OK. Após este processo, todos os campos apareceram na aba Data, presente na janela Data Tree, localizada na lateral direita do editor. A **Figura 2** apresenta a janela Data Tree com os dados adicionados.

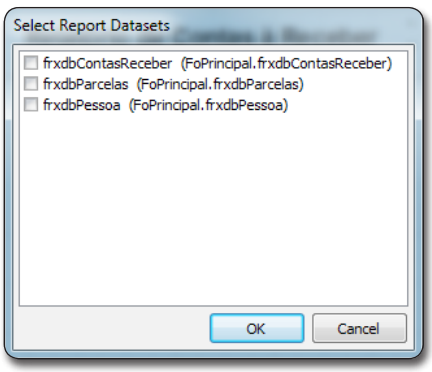

Figura 1. Janela para a integração dos datasets

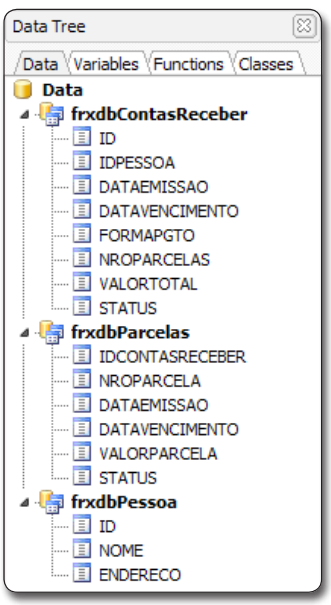

Figura 2. Janela Data Tree com os campos dos DataSets

A partir deste momento, o popular recurso "arrastar e soltar" pode ser empregado para que os campos sejam rapidamente inseridos em seus devidos recipientes no relatório. Para dar continuidade ao projeto, devem ser adicionadas as bandas referentes a cada DataSet. A primeira que deve ser adicionada é a do tipo Master Data. Ao adicionar tal banda, o desenvolvedor também deve informar a qual DataSet a mesma está relacionada, através da janela que é aberta automaticamente, denominada "Select DataSet". Neste exemplo ela tem uma ligação com o objeto frxdbPessoa. Posteriormente, uma banda do tipo Detail Data também deve ser adicionada e relacionada com o objeto frxdbContasReceber. Um ponto extremamente importante na elaboração de relatórios com o FastReport é com relação ao posicionamento das bandas, ou seja, uma banda do tipo Master Data, por exemplo, deve ser inserida antes de uma Detail Data e não o inverso, o que pode ocasionar um erro. Antes de inserir a última banda para os dados será adicionada uma banda do tipo Group Header, com o intuito de agrupar e empregar algumas funcionalidades. Ao adicionar esta banda será aberta a janela "Group", onde o desenvolvedor deve informar qual é o Dataset e o campo que será adotado para o agrupamento. A **Figura 3** apresenta a janela "Group" com a definição do campo de agrupamento.

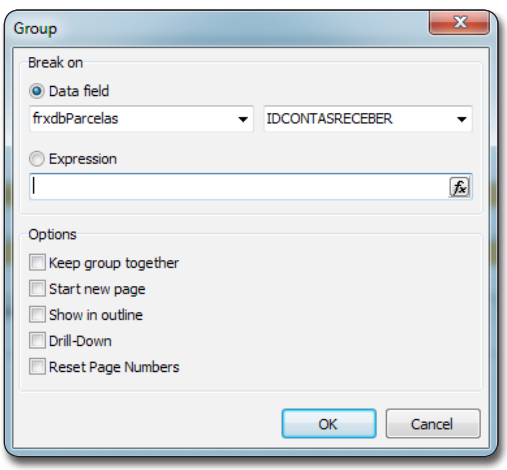

Figura 3. Janela "Group" para a definição do agrupamento

Como pode ser observado na **Figura 3**, o DataSet selecionado foi o frxdbParcelas e o campo de agrupamento foi definido como IDCONTASRECEBER.

Neste momento, os campos dos DataSets presentes na janela Data Tree devem ser arrastados para seus devidos recipientes (DataBands), na ordem desejada para a exibição das informações. Após a adição dos mesmos foi adicionada uma banda do tipo Group Footer, para o Subdetail Band, que por sua vez, recebeu uma linha divisória através da opção Line Object presente no botão Draw, o último botão da barra de ferramentas lateral do editor. Também foram incluídas linhas no cabeçalho. Além disso, foi adicionada uma banda do tipo Report Summary, cujo intuito é apresentar uma informação geral resumida do relatório, ou em outras palavras, pode ser utilizada para demonstrar o valor total ou número máximo de registros e assim por diante, exibindo tais dados no final do relatório. Para este exemplo, foi adicionado um objeto do tipo System Text em tal banda. A **Figura 4** apresenta a janela que é exibida ao inserir um objeto System Text.

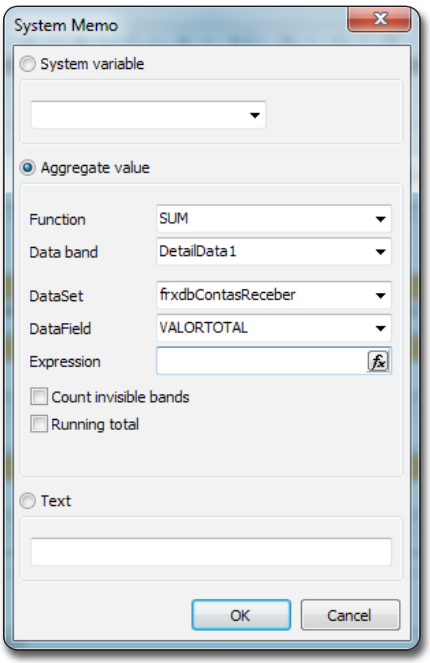

Figura 4. Janela "Sytem Memo" para adição de campos especiais

A janela do objeto System Text possui diversas funcionalidades. Através da mesma o desenvolvedor pode inserir informações úteis automaticamente. Em System Variable, existe a possibilidade de adicionar facilmente ao relatório a hora atual, a data, o número da página atual, o total de páginas, entre outros. Através de Aggregate Value, que foi o empregado neste exemplo, é possível utilizar facilmente uma função como SUM, MAX, COUNT, entre outras e definir qual a banda, DataSet e campo que deve ser considerado para o cálculo, onde dentro deste contexto, será calculado o valor total das contas de acordo com o DataSet frxdbContasReceber e o campo VALORTOTAL.

Por fim, também existe a opção Text onde o desenvolvedor pode elaborar uma expressão ou simplesmente um texto qualquer. Como este objeto representa um valor monetário, o mesmo também recebeu uma configuração especial para sua exibição. Através da propriedade DisplayFormat, é possível definir uma formatação para a visualização do campo em questão. Sendo assim, a sub-propriedade "Kind" foi definida como fkNumeric, e a propriedade FormatStr recebeu o parâmetro "%2.2m", utilizado para apresentação monetária, por último o DecimalSeparator foi definido com o caractere ",". Este processo também foi efetuado para os outros objetos relativos aos campos de valores dos DataSets frxdbContasReceber e frxdbParcelas (VALORTOTAL e VALORPARCELA).

Outras propriedades dos objetos que também foram alteradas no desenvolvimento deste relatório foram a Align e HAlign. Em Align o desenvolvedor define o alinhamento para determinado

objeto com relação ao relatório, como por exemplo, o título centralizado através do valor center e em HAlign, o alinhamento interno do objeto, como por exemplo, o alinhamento dos valores à direita (right). Vale à pena ressaltar também que quando um campo do DataSet é arrastado diretamente para uma banda, o FastReport cria um objeto do tipo Texto, porém com um valor pré-definido para o mesmo através dos caracteres "[ ]", como por exemplo, [frxdbPessoa."ID"]. Sendo assim, se o desenvolvedor desejar adicionar um texto antes da exibição da informação contida no campo, basta simplesmente dar um duplo clique sobre o objeto e digitar a informação desejada seguida pela expressão que já está preenchida, como por exemplo, "ID: [frxdbPessoa."ID"] ".

O FastReport também conta com um recurso interessante, presente nas bandas do tipo GroupHeader, que é denominado DrillDown. Através do mesmo, o desenvolvedor pode adicionar uma funcionalidade para o usuário que permite exibir/ocultar as informações que pertencem a tal grupo definido, através de um simples clique em qualquer lugar do mesmo. Esta funcionalidade permite que relatórios comuns, como sintético e analítico, possam ser feitos de uma única vez e em um único relatório. Sendo assim, na banda GroupHeader1 adicionada anteriormente, o valor da propriedade DrillDown foi alterado para True. Também foi adicionado um objeto do tipo TextObject que recebeu o texto "EXIBIR/OCULTAR PARCELAS", apenas para apresentar de certa forma esta opção ao usuário.

Finalizado este processo, o relatório encontra-se concluído. Um ponto importante que deve ser ressaltado nesta etapa é com relação à sua gravação. Como mencionado anteriormente, quando um componente do tipo TfrxReport é adicionado ao formulário, o relatório fica embutido no arquivo DFM do Delphi, o que implica que o mesmo não necessita ser obrigatoriamente gravado no editor, mas pode ser salvo diretamente no Delphi, onde ficará posteriormente dentro do executável da aplicação. Contudo, se o desenvolvedor desejar reaproveitar o relatório em outros projetos ou para outras finalidades, ele pode gravá-lo diretamente em um arquivo no disco rígido, podendo inclusive carregá-lo em tempo de execução através do método mencionado anteriormente, Load FromFile. A **Figura 5** apresenta o visual final do relatório através do Editor do FastReport.

Concluída a elaboração do relatório, a próxima etapa é definir um simples trecho de código no formulário da aplicação para efetuar seu carregamento. Para tanto, foi adicionado um objeto do tipo TSpeedButton. A **Listagem 2** apresenta o código atribuído ao evento OnClick do componente TSpeedButton inserido.

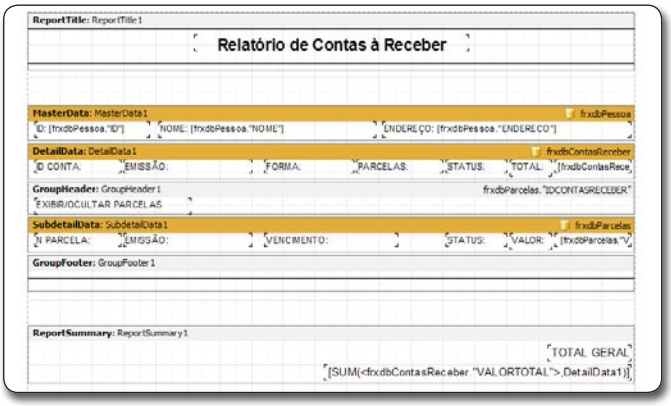

Figura 5. Visual final do relatório no editor do FastReport

Listagem 2. Código do evento OnClick do TSpeedButton

```
procedure TFoPrincipal.btExibirRelatorioClick(Sender: TObject);
begin
  if not SQLConnection1.Connected then
    SQLConnection1.Connected := True;
  sdsPessoa.Open;
  sdsContasReceber.Open;
  sdsParcelas.Open;
  frxReport1.ShowReport(True);
end;
```

É possível notar neste trecho de código que em um primeiro momento, é efetuada uma validação para verificar se a conexão com o banco de dados está aberta (SQLConnection1.Connected) e caso não esteja, a mesma é efetuada. Em seguida, todos os DataSets são abertos para o carregamento das informações com a utilização do método Open. Para concluir, o método ShowReport do objeto frxReport1 é chamado com o parâmetro ClearLastReport definido como True. Este parâmetro é utilizado para limpar um relatório que tenha sido previamente construído, se este for o caso. Como mencionado, o método ShowReport faz com que a janela de pré-visualização seja exibida e o relatório possa ser impresso.

Neste momento o exemplo está concluído. A **Figura 6** apresenta o visual final do relatório exibido através da aplicação com algumas informações para o teste do mesmo.

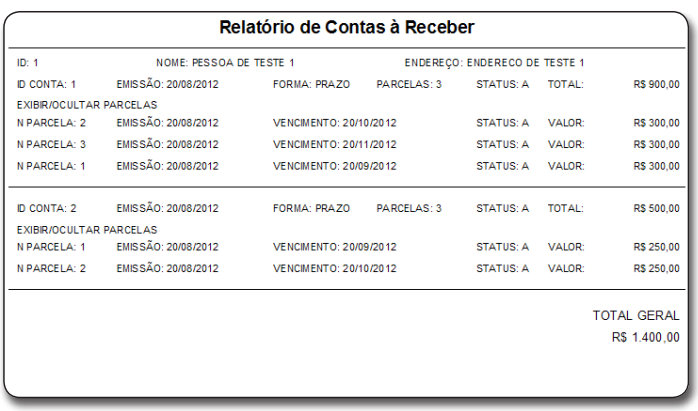

Figura 6. Visão do relatório em execução

Conclusão

É indiscutível a importância da apresentação de relatórios pelos Softwares das organizações. Diante deste fator, ter a disponibilidade de ferramentas com diversos recursos que permitem a elaboração de relatórios de uma forma rápida e simples é extremamente importante, e uma das principais ferramentas dentro deste contexto é o FastReport. Com o editor próprio do mesmo, o desenvolvedor consegue ter um aumento na produtividade, uma vez que recursos como "arrastar e soltar" e funções prontas estão disponíveis de uma forma bem prática. Além disso, a possibilidade de gravar o relatório em um arquivo é interessante ao que diz respeito à sua reutilização em outros projetos semelhantes.

Outro ponto forte do FastReport é sua variedade de opções para a exportação do relatório, facilitando para o usuário final o envio através de um meio eletrônico como e-mail, ou ainda ter a possibilidade de exportar o relatório para uma planilha e assim poder efetuar filtros e outros cálculos que desejar.

Diante de tantas vantagens, investir na aprendizagem desta ferramenta é um fator que vale à pena. Até mesmo cogitar uma migração para futuros projetos se torna uma opção interessante, uma vez que é estável e tem evoluído de acordo com as necessidades da atualidade.

Nesta primeira parte foi apresentada uma visão geral e básica do FastReport para a elaboração de relatórios, apresentando um exemplo baseado em três níveis de DataSets (Master/Detail/Subdetail). No próximo artigo, serão abordadas outras funcionalidades, como variáveis, exportação do relatório, preenchimento de informações em tempo de execução, entre outros.

Giuliano Scombatti Pinto
giuliano@sygnux.com.br
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Site do FastReport
http://www.fast-report.com

Download do FastReport para FireMonkey
http://www.fast-report.com/en/download/fast-report-firemonkey.html

DevSpace do Autor
http://www.devmedia.com.br/space.asp?id=283274

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Seu **conhecimento vale muito** na DevMedia

Publique artigos e vídeos gratuitamente no site e participe do programa **PayContent.**

Sua participação pode ser premiada: quanto mais você postar, mais chances de ganhar.

Para saber mais acesse:
www.devmedia.com.br/paycontent

DevMedia

Padrões de projeto no dia a dia

Alguns exemplos de uso

Resumo DevMan

De que se trata o artigo:

Este artigo trata da aplicação dos padrões de projeto e suas boas práticas em projetos já existentes, em situações comuns que podem ser reescritas de forma orientada a objetos, mostrando que mesmo em poucas partes pode-se obter flexibilidade.

Em que situação o tema é útil:

Quando se deseja ter partes de um projeto que possam responder de forma mais rápida às mudanças de requisitos dos clientes, seguindo boas práticas conceituadas.

Padrões de projeto no dia a dia :

Saber como aplicar uma teoria da orientação a objeto é a dúvida de todos que estão começando ou estudando o assunto. Nesse artigo vamos abordar os conceitos que formam a base dos padrões de projeto e como podemos aplicar os padrões em trechos de código já existente.

Por mais que sempre se fale de padrões de projeto, ou da própria orientação a objetos, sempre existem aqueles que dizem: "isto não é para mim", ou "não consigo aplicar, meu projeto não está totalmente desenvolvido com OO".

Engana-se quem acredita que não é possível seguir algumas boas práticas só por não ter todo o sistema desenvolvido de forma orientada a objetos. É possível sim aplicar em pequenas partes algumas boas práticas e padrões. Como fazer isso? É justamente o que esse artigo traz. Mas antes, uma visão do que está por de trás dos padrões.

Princípios

Existe um fato irrefutável no desenvolvimento de sistemas: requisitos sempre mudam. Outro fato que sempre vejo nas organizações é que elas sabem disso e por isso, tentam melhorar sua forma de captação desses requisitos. Isso é bom. Contudo, a atenção não deve estar presa somente a isso, não significa que por se ter requisitos de qualidade, teremos um sistema de qualidade ou, se esses requisitos modificarem será fácil aplicar a modificação pedida.

É aí que a orientação a objetos em conjunto com os padrões pode ajudar. Existem alguns princípios de modelagem e programação OO que levam os padrões de projetos a serem considerados como boas práticas, esses mesmos conceitos podem ser aplicados sem a utilização dos padrões. Portando, a ideia é entender esses princípios e procurar aplicá-los em nosso desenvolvimento. São eles: Modelar para interfaces, favorecer composição sobre herança e encontrar o que varia, e encapsulá-lo.

Modelando, desenvolvendo para interfaces

Em qualquer desenvolvimento, seja ele orientado a objetos ou não, temos dois problemas que impedem do mesmo ser fácil de ser alterado: baixa coesão e alto acoplamento.

A baixa coesão é quando encontramos rotinas/classes que realizam várias funções, mais do que deveriam fazer por seu conceito. Já o alto acoplamento é quando temos rotinas/classes que são muito dependentes de outras, aonde um alteração nessa conduz a um cascateamento de alterações, ou seja, é o quanto as rotinas/classes estão dependentes uma das outras.

Mas porque uma interface resolveria esse problema? Uma interface ou classe abstrata simplesmente expõem uma assinatura de métodos que pode ser utilizada por outra rotina. A implementação real não está nela, mas sim outra classe que a implementa. Dessa forma, se esperarmos por uma interface, teremos acesso somente à definição dos métodos, como eles serão implementados não será importante. Mais adiante teremos a parte prática onde será possível ver isso acontecendo.

Compor ou herdar?

Quem está aprendendo orientação a objetos sempre se atenta muito ao recurso de herança. É a primeira coisa que estudantes aprendem. É a primeira forma de realizar a tão sonhada reutilização de funcionalidades, isso porque ela permite que o interior de uma classe base seja vista pelas suas subclasses. Esse é o modo mais fácil.

Existe um segundo recurso para reutilização de funcionalidade, a composição. Nela objetos diferentes são interligados através de suas interfaces para garantir reutilizações mais complexas. A composição pode ser um pouco mais difícil de entender inicialmente, mas possui vantagens sobre a herança.

A herança permite que de forma fácil seja alterada a implementação de uma funcionalidade. Isso é obtido através da técnica de sobrecarga de operações. Por ser fácil de implementar, passa uma ideia de que agora aquele trecho de código está flexível, mas não é verdade. Ela é resolvida em tempo de compilação e isso deixa o sistema um pouco inflexível porque em tempo de execução qualquer alteração na implementação de uma classe base não é permitida. Os mais puristas em orientação a objetos dizem que a principal desvantagem da herança está na exposição do interior de uma classe base à suas subclasses, quebrando assim o conceito de encapsulamento. Essa exposição conduz a um modelo de dependência de implementações entre as classes envolvidas, isso porque a parte exportada da classe base faz parte também das subclasses.

Qualquer alteração na classe base implica em alterações nas subclasses, o que não é recomendado quando se deseja um modelo flexível.

A composição por sua vez é resolvida em tempo de execução. Ela exige que as interfaces sejam respeitadas, consequentemente, devem ser bem planejadas para que não haja impedimentos de reutilização, isso porque a implementação de composição é feita através de referências.

Algumas desvantagens vistas na herança, como quebra de encapsulamento e dependência de implementação, não são vistas na composição, isso porque os objetos envolvidos são acessados somente por suas interfaces.

O impacto disso na modelagem, é que classes manterão seu encapsulamento e estarão voltadas para sua tarefa apenas, porque não há o que herdar de outras classes, obedecendo assim o conceito de alta coesão.

Porém, como nada é perfeito, o uso de composição prende o comportamento do modelo aos relacionamentos entre os objetos, que pode implicar em maior complexidade. Os padrões de projeto fazem uso extenso de composição e quando é necessário criar heranças, elas são baseadas em classes abstratas. Esse é um dos motivos que consagram os padrões de projeto como prática de flexibilização de modelos, deixando um código preparado para eventuais mudanças de requisitos.

Encapsular o que varia

O encapsulamento sempre é visto como uma forma de ocultação de informações, mas ele é mais que isso. Podemos encapsular implementações, comportamentos e classes derivadas. A ideia é pensar no que queremos deixar flexível no sistema, no que queremos poder mudar sem remodelar e encapsular isso. Para deixar isso mais claro, vamos pensar numa situação comum. Digamos que temos em nosso sistema uma classe (TCompra) que representa uma compra efetuada, mas essa mesma compra pode ser paga de diversas formas. O requisito é que cada compra saiba como que foi paga. Como poderíamos implementar isso?

A forma mais simples e rápida seria ter em TCompra um campo que indicasse qual foi a forma de pagamento utilizada, esse campo seria uma espécie de flag. Em um primeiro momento não vemos nada de errado nisso. Contudo, ao decorrer do desenvolvimento percebemos que sempre que for necessário ter que obter alguma informação do pagamento, é necessário checar o valor desse flag. Isso não é bom.

Uma segunda forma de implementação seria algo mais orientado a objetos. Teríamos para cada tipo de forma de pagamento, um tipo de subclasse de TCompra. Esse pode ser considerado um detalhamento excessivo, porque sempre que for necessário utilizar a classe TCompra, é preciso também ter conhecimento de suas subclasses, pois é nelas que temos a informação da forma de pagamento. Vamos tentar aplicar agora o que foi sugerido, encapsular o que varia. Pensando nisso, o que está variando no contexto? A forma de pagamento. Assim podemos extrair o conceito "Forma de Pagamento" em uma classe abstrata e em TCompra ter um campo que referencia essa forma de pagamento. Com isso se cria uma composição. Agora, para cada tipo de forma de pagamento, teríamos uma implementação da classe abstrata. TCompra não precisa saber quantas variações existem, ela conhece apenas a interface da classe abstrata, que é o suficiente, já que essa interface é obedecida por todos os tipos variantes de forma de pagamento. Assim, definimos que:

- Classe abstrata: É uma abstração que conceitua todas as variações de uma classe;
- O que é comum: Define classes abstratas;
- Variações: Definem as variações das classes abstratas, sua implementação;
- Especificação: Definição de interfaces que permita o uso e atenda as variações.

Padrões de projeto

Até o momento o que foi apresentado pode ser considerado como base para os padrões de projeto e que não dependem dos mesmos para ser implementado. Um padrão é uma ideia que tem sido útil em um contexto prático e que provavelmente será útil em outros, ou seja, é uma solução já testada para problemas de modelagem que acontecem sempre. Eles foram organizados nas seguintes categorias: criacionais, estruturais e comportamentais. Os criacionais abstraem a instanciação e o processo de instanciação de objetos, de tal forma que um sistema pode não depender (saber) como são criados determinados objetos. Para conseguir essa independência e flexibilidade, os padrões criacionais encapsulam as classes concretas que existem. Os estruturais flexibilizam a criação de classes e objetos, permitindo que classes independentes trabalhem juntas. Isso é conseguido através do uso de herança para compor interfaces ou implementações. Já os comportamentais gerenciam algoritmos e responsabilidades de objetos.

Cada categoria dessas possui vários padrões definidos. Não serão relacionados todos aqui, mas será mostrado como aplicar alguns em seus sistemas já existentes.

Tutorial

Situação 1: Leitura de arquivos

Quando existe entrada de dados de forma automatizada, essa geralmente acontece por meio da transmissão de arquivos. É muito comum lermos um arquivo hoje, em um determinado formato e amanhã, por alguma situação comercial, termos que ler um arquivo diferente para a mesma rotina. Uma situação comum ocorre quando numa determinada rotina do sistema le-se um arquivo texto onde os dados estão organizados por colunas (por exemplo: da posição 1 até 45 representa o nome de uma pessoas, da 46 à 49 seu CPF) e, dependendo do cliente, essa mesma informação também pode vir através de um arquivo XML. A pergunta aqui é: como organizar o código de modo que fique fácil incluir um novo formato de leitura, caso isso aconteça? Resposta: usando um padrão de projeto.

Existe um padrão chamado Strategy. Seu objetivo é bem simples: definir uma família de algoritmos, encapsular cada um e torná-los intercambiáveis, ou seja, poder em um momento usar um, em outro momento outro, sem alterar a estrutura principal. Isso acontece quando isolamos a abstração em uma interface e ocultamos as variações em classes derivadas.

Para demonstrar seu funcionamento vou mostrar a implementação do meu caso real citado anteriormente.

Nota

Será exibido código somente referente aos padrões, contudo, o código completo está disponível para download.

Neste aplicativo de exemplo (**Figura 1**) temos um TClientDataSet que irá armazenar os dados de alguns clientes. Esses dados ora são recebidos no formato TXT, ora XML. Nesse contexto, o aplicativo assume que existe em sua pasta de execução um arquivo chamado clientes.txt ou clientes.xml. A rotina de leitura foi centralizada em uma classe chamada TArquivoClientes (**Listagem 1**). Isso foi feito para isolar a lógica em relação ao formulário. O formulário não sabe nem qual arquivo deve chamar. Se essa pequena lógica estivesse escrita dentro do código do formulário, isso não seria reutilizável, como está separada, podemos reutilizar.

Ao observar a seção pública da classe, temos a criação de um construtor que recebe como parâmetro um tipo enumerado chamado TTipoArquivo. Esse dado foi construído com a finalidade de representar os tipos de arquivos existentes, como é mostrado a seguir:

```
TTipoArquivo = (tpTXT, tpXML);
```

Muitos desenvolvedores não conhecem esse tipo de dado e acabam utilizando o que chamamos de "valores mágicos", que nada mais são do que convenções que ninguém consegue identificar no meio do código, por exemplo, definir uma variável inteira que armazene 0 (zero) quando for TXT e 1 (um) quando for XML. Então, outro desenvolvedor pode olhar esse código e se perguntar "o que é esse número 1 aqui?". Percebem o problema?

Listagem 1. Classe TArquivoClientes

```
01 TArquivoClientes = class
02 private
03   FTipoArquivo: TTipoArquivo;
04   FDataSetClientes: TClientDataSet;
05   FLeitura: TEstrategiaLeitura;
06   procedure SetDataSetClientes(const Value: TClientDataSet);
07 public
08   constructor Create(tipoArquivo: TTipoArquivo); reintroduce;
09   destructor Destroy; override;
10   property DataSetClientes:TClientDataSet read FDataSetClientes
       write SetDataSetClientes;
11   procedure LerArquivo;
12 end;
```

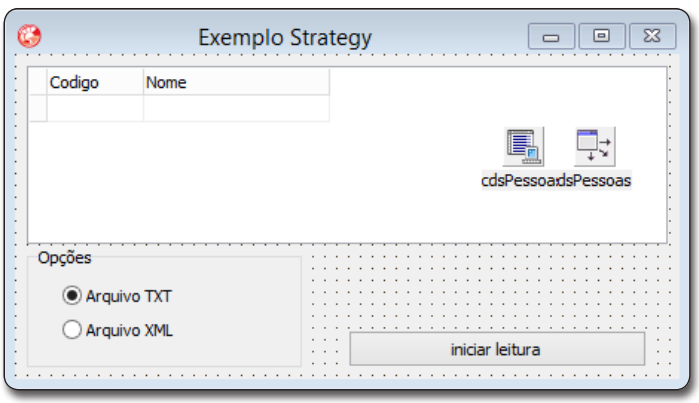

Figura 1. Tela do exemplo de Strategy

Em Delphi, todo objeto possui um construtor implícito, o Create, que por padrão não recebe parâmetro algum. Contudo, em nossa lógica queremos forçar o usuário a informar no momento da criação de um objeto TArquivoClientes qual o tipo de arquivo será lido, isso é obtido com o uso da palavra reservada **reintroduce**. Ela faz com que o construtor original seja substituído pelo nosso construtor, assim, até mesmo no recurso de auto completar do editor do IDE apenas nossa versão será exibida.

Temos também uma propriedade do tipo TClientDataSet, que irá apontar para o que existe no formulário. Na linha 5 temos a definição de um campo do tipo TEstrategiaLeitura. Esta é uma classe abstrata que foi criada para encapsular o que varia, conforme boa prática citada anteriormente e conforme estrutura do próprio padrão Strategy (**Figura 2**).

Ao fazer um comparativo com **Figura 2**, podemos dizer que Client é o formulário da aplicação, é um cliente que simplesmente faz uso de um objeto de contexto, representado no nosso exemplo pela classe TArquivoClientes. O contexto por sua vez agrega uma referência a um objeto abstrato/interface, aqui chamado de TEstrategiaLeitura (**Listagem 2**).

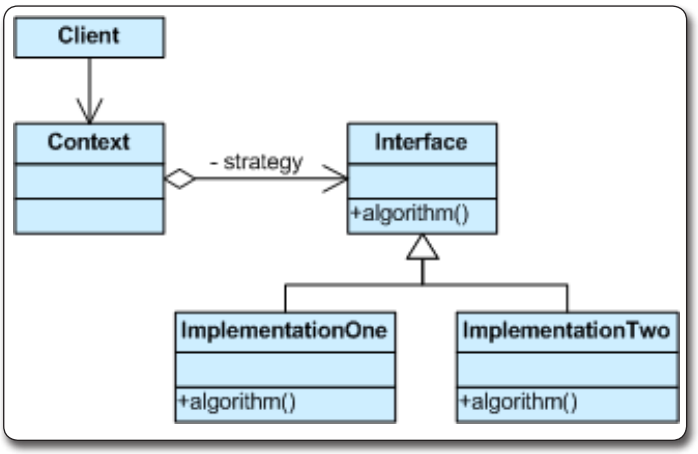

Figura 2. Estrutura clássica do padrão Strategy

Essa classe define os únicos métodos que TArquivoClientes conhecem, e como estão marcados com as palavras reservadas **virtual** e **abstract**, devem ser sobrecarregados por classes descendentes. Aqui já vemos certa flexibilidade, pois o objeto de contexto (que é utilizado pelo formulário) conhece apenas a classe abstrata, ele não conhece nenhuma das possíveis implementações. E falando nelas, elas foram implementadas na classe TLeituraTexto (**Listagem 3**) e TLeituraXML (**Listagem 4**).

Listagem 2. Classe abstrata TEstrategiaLeitura

```
TEstrategiaLeitura = class
protected
  procedure AbrirArquivo; virtual;abstract;
public
  procedure IniciarLeitura(umDataSet: TClientDataSet); virtual; abstract;
end;
```

Listagem 3. Classe TLeituraTexto

```
TLeituraTexto = class(TEstrategiaLeitura)
  private
    FArquivo: TStringList;
  protected
    procedure AbrirArquivo; override;
  public
    constructor Create;
    destructor Destroy; override;
    procedure IniciarLeitura(umDataSet: TClientDataSet); override;
end;
```

Listagem 4. Classe TLeituraXML

```
TLeituraXML = class(TEstrategiaLeitura)
  private
    FArquivo: TXMLDocument;
  protected
    procedure AbrirArquivo; override;
  public
    constructor Create;
    procedure IniciarLeitura(umDataSet: TClientDataSet); override;
end;
```

Ao observar as duas classes, vemos que ambas herdam de TEstrategiaLeitura, como era o esperado. E também, ambas sobrecarregam os métodos definidos na classe pai, através da palavra reservada **override**. Agora que temos as peças do quebra cabeça, vamos analisar como tudo está ligado, começando do formulário.

Nele temos um campo privado que armazena o tipo de arquivo que foi escolhido (via TRadioButton). Ao clicar no botão "iniciar leitura" executamos o nosso contexto, TArquivoClientes, como mostra a **Listagem 5**.

Listagem 5. Executando TArquivoClientes

```
procedure TForm1.btnLeituraClick(Sender: TObject);
begin
  FLeitura := TArquivoClientes.Create(FTipoArquivo);
  FLeitura.DataSetClientes := cdsPessoas;
  FLeitura.LerArquivo;
  FreeAndNil(FLeitura);
end;
```

É importante observar que o formulário não está diretamente amarrado a nenhum tipo de leitura, temos apenas um campo armazenando qual o tipo de arquivo será lido. Toda a lógica está contida em TArquivoClientes e posteriormente delegada às classes de implementação de leitura em si. Passamos ao objeto FLeitura o que é pedido e executamos seu método LerArquivo. O construtor de TArquivoClientes simplesmente decide qual implementação de leitura será utilizada, essa é a "estratégia" que dá nome ao padrão, como visto a seguir na **Listagem 6**.

Listagem 6. Construtor de TArquivoClientes

```
constructor TArquivoClientes.Create(tipoArquivo: TTipoArquivo);
begin
  FTipoArquivo := tipoArquivo;
  case FTipoArquivo of
    tpTXT: FLeitura := TLeituraTexto.Create ;
    tpXML: FLeitura := TLeituraXML.Create ;
  end;
end;
```

Lembre-se que FLeitura é do tipo TEstrategiaLeitura e ambas as implementações herdam desse tipo, assim, elas podem se passar por um objeto do tipo TEstrategiaLeitura, que é exatamente o que acontece nesse código. Isso mostra o polimorfismo em ação.

Quando o formulário executa o método LerArquivo, este encaminha a execução para a estratégia utilizada em FLeitura:

```
FLeitura.IniciarLeitura(DataSetClientes);
```

Olhando de uma forma geral, escrevemos bastante código apenas para a leitura de dois tipos de arquivo. Precisamos realmente de tudo isso? Talvez ao comparar com alguma implementação que você leitor tenha, possa achar que não precisamos disso tudo,

bastaria dar um duplo clique ali mesmo no botão do formulário, fazer um "if" para verificar qual TRadioButton está marcado e implementar a leitura dos arquivos baseados nisso.

Contudo, a solução proposta separa as responsabilidades, permitindo o reuso de recursos e, se algum tipo novo de arquivo surgir, por exemplo, o formato XLS, bastaria apenas incluir esse novo tipo no enumerador TTipoArquivo, criar mais um TRadioButton que o representasse, criar uma classe concreta que herde de TEstrategiaLeitura e no construtor de TArquivoClientes, adicionar esse novo tipo ao Case utilizado. Agora pense como seria a implementação de um novo tipo de leitura no código amarrado ao formulário. Teríamos mais um "if" a verificar. Falando em "if", na solução proposta com o padrão de projeto não temos o uso de nenhum, você percebeu?

Situação 2: Instanciar objetos

Muitas vezes temos que instanciar via código, vários componentes e configurá-los. Um exemplo real e simples é a criação em tempo de execução de componentes de acesso a dados, que precisam ser configurados sempre. Tenho um sistema que utiliza os componentes IBX e sempre que preciso criar um objeto TIBQuery, preciso apontar para ele quem é o objeto de conexão (TIBConnection) e o de transação (TIBTransaction).

Então, em um procedimento muito comum fazemos conforme a **Listagem 7**.

Aparentemente não há nada de errado com o código, contudo, pensando que em todo lugar que for necessário instanciar um TIBQuery é necessário referenciar o DataModule1, não parece certo. E se por algum motivo o container dos objetos de conexão e transação não for mais o DataModule1? Será necessário então varrer todo o código corrigindo as referências.

Entre os padrões criacionais, que tratam exatamente de problemas para criar objetos, existe o Factory Method. Ele é útil quando é necessário encapsular os detalhes para a criação de objetos de tal forma que uma classe consumidora (a classe que irá instanciar o objeto) não necessite saber desses detalhes. A **Figura 3** exibe sua estrutura formal.

Listagem 7. Criação de um TIBQuery

```
Var
  Qry: TIBQuery;
Begin
  Qry := TIBQuery.Create;
  Qry.Connection := DataModule1.ObjetoConexao;
  Qry.Transaction := DataModule1.ObjetoTransacao;
  //continua o uso
End;
```

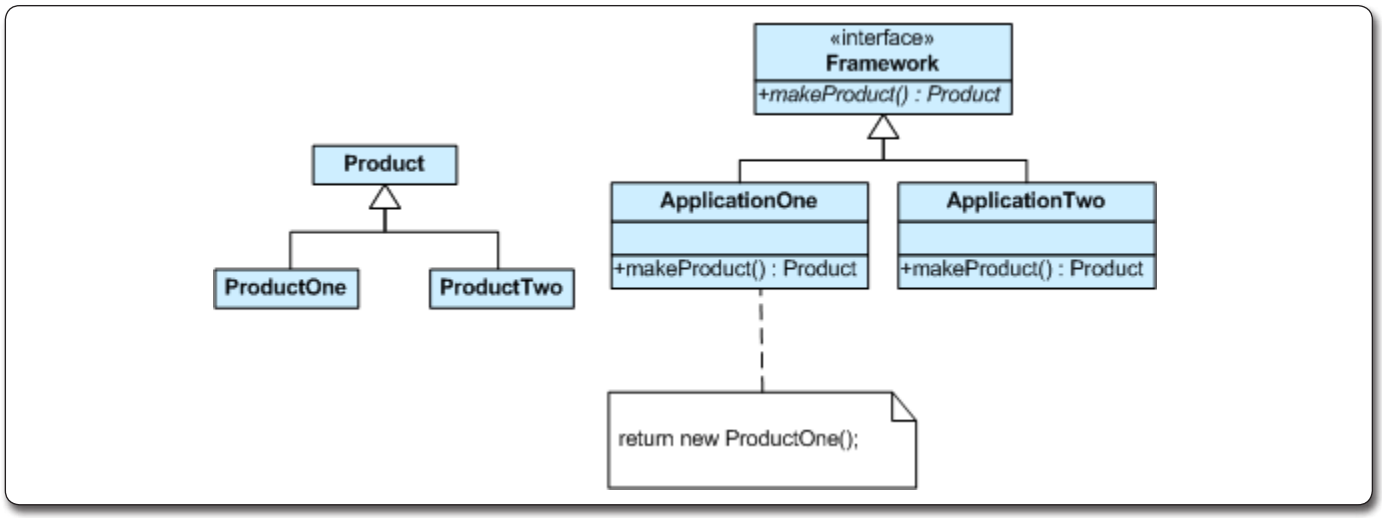

Figura 3. Estrutura do padrão Factory Method

Ele é composto por uma classe abstrata/interface que expõe um método que irá retornar o objeto pronto para uso. Essa classe abstrata é então implementada por uma classe que realmente escreve o código que instancia o requerido objeto, já configurado. A solução para o problema mostrado na **Listagem 7** está na **Listagem 8**.

Listagem 8. Implementação do Factory Method

```
TIBQueryFactory = class
 public
  class function CriarQueryComum: TIBQuery ;
 end;

class function TIBQueryFactory.CriarQueryComum: TIBQuery;
var
 qry: TIBQuery;
begin
 qry := TIBQuery.Create(nil);
 //Aqui se configuraria o TIBQuery
 //qry.DataBase := DataModule1.ObjetoConexao;
 //qry.Transaction := DataModule1.ObjetoConexao;
 Result := qry;
end;
```

A classe TIBQueryFactory centraliza os detalhes para se criar um novo objeto TIBQuery. Ela expõe o método de classe (estático, que dispensa a instanciação) CriarQueryComum que retorna o novo objeto já configurado. Caso seja necessário um objeto TIBQuery Factory com outro tipo de configuração, basta criar outro método de criação, por exemplo, um CriarQueryAlternativa, que também irá encapsular os detalhes. Agora, quando for necessário um novo TIBQuery, utilizamos a factory (fábrica) criada, como mostra a **Listagem 9**. O download deste exemplo também está disponível.

Agora se for necessário mudar de DataModule1 para DataModule2, o impacto é menor, visto que basta corrigir a classe Factory e não mais todo lugar onde se instanciava um objeto TIBQuery.

Listagem 9. Utilizando TIBQueryFactory

```
procedure TForm2.Button1Click(Sender: TObject);
var
 qryTemporaria: TIBQuery;
begin
 qryTemporaria := TIBQueryFactory.CriarQueryComum;
 //continua o uso
end;
```

Situação 3: Mediação

O padrão de projeto Mediator define um objeto que encapsula a interação de um conjunto de objetos. Isso promove o baixo acoplamento, impedindo que objetos sejam referenciados de forma explícita por um objeto cliente, de tal forma que podemos variar essa interação. A ideia é ter um objeto intermediário que desacopla muitas partes.

A intenção desse padrão ficou confusa? Saiba que utilizamos uma implementação dele constantemente ao fazer uso das caixas de diálogo disponíveis na unit Vcl.Dialogs.

Uma caixa de diálogo representa uma coleção de controles, como botões, caixas de seleção, campos, etc. Existem dependências entre esses controles. Por exemplo, um botão desabilitado muda seu estado para habilitado quando o texto em um campo muda. Selecionar o item em uma lista de opções pode mudar o conteúdo de um campo. Os mais diversos diálogos possuem diferentes dependências. Para controlar tudo isso a VCL implementou o padrão Mediator em um componente e esse é responsável por controlar e coordenar as interações do grupo de objetos que ele contém. Uma situação real que acontece comumente é referente a envio de e-mails. Dada uma situação onde é preciso enviar um e-mail para um determinado cliente ou clientes que estão em um ClientDataSet. A rotina de envio de e-mail é uma procedure que recebe como parâmetro um endereço de destinatário. De uma forma simplista, poderíamos utilizar dentro da procedure uma referência ao ClientDataSet e lá mesmo percorrer os destinatários.

Essa implementação está errada? Não, está com erros de arquitetura. Ao fazer isso, cria-se uma dependência entre a rotina de envio e o ClientDataSet, aumentando o acoplamento. A alternativa é fazer um mediador, que se encarregue de enviar à procedure os e-mails destinatários. Na **Listagem 10** seu código é apresentado.

Listagem 10. Mediador de envio de e-mails

```
TEnvioEmailMediator = class
 procedure EnviarEmail(cds: TClientDataSet);
end;

procedure TEnvioEmailMediator.EnviarEmail(cds: TClientDataSet);
begin
  cds.First;
  while not cds.Eof do
  begin
   MediatorU.EnviarEmail(cds.FieldByName('email').AsString);
   cds.Next;
  end;
end;
```

A classe TEnvioEmailMediator expõe um único método nesta situação, EnviarEmail, que recebe como parâmetro um objeto do tipo TClientDataSet. Esse é o cenário projetado, como dito, os clientes estão em um TClientDataSet. A implementação desse método é simples. Ela se encarrega de percorrer os clientes passados como parâmetro e redireciona para a procedure EnviarEmail o endereço de destino. O importante aqui é entender que TEnvioEmailMediator é que está orquestrando o relacionamento entre a procedure e o TClientDataSet real. Com isso pronto, quando for necessário enviar e-mails utilizamos o mediador, como mostra a **Listagem 11**.

A classe cliente, que usa o mediador, está totalmente alheia ao relacionamento existente, sua lista de clientes e o método Enviar Email. Tudo foi simplificado e desacoplado, conforme o objetivo desse padrão.

Listagem 11. Mediador em uso

```
procedure TForm3.Button1Click(Sender: TObject);
var
 mediator : TEnvioEmailMediator;
begin
 mediator := TEnvioEmailMediator.Create;
 mediator.EnviarEmail(cdsPessoas);
 FreeAndNil(mediator);
end;
```

Conclusão

É possível melhorar a qualidade de código, inserindo nele partes que seguem boas práticas da orientação a objetos, como os padrões de projeto. As situações apresentadas aqui são corriqueiras, e a grande maioria dos desenvolvedores possui algo parecido. E mesmo não sendo aplicações totalmente orientadas a objetos, puderam ser aprimoradas.

Fica o desafio agora para você leitor, procure em seu código o que pode ser melhorado e não tenha medo, melhore.

Paulo Quicoli
pauloquicoli@gmail.com
Editor da revista ClubeDelphi e editor técnico .Net Magazine. Formado em processamento de dados pela FATEC-TQ. Atua como analista programador de sistemas e coordenador técnico da equipe de desenvolvimento .NET da Siplan Control-M (www.siplancontrolm.com.br) unidade Jaboticabal, e professor da FATEC, campus Taquaritinga. Blog: http://quicoli.worpress.com

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Somos tão apaixonados por tecnologia que o nome da empresa diz tudo.

Porta 80 é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia.
Somos uma equipe composta de gente que entende e gosta do que faz,
assim como você.

Estrutura

100% NACIONAL. Servidores de primeira linha, links de alta capacidade.

Suporte diferenciado

Treinamos nossa equipe para fazer mais e melhor. Muito além do esperado.

Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.

1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única. Conheça!

Porta 80
WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce | Aplicações | Streaming | Email corporativo

porta80.com.br | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486

TENHA ACESSO ILIMITADO A TODO CONTEÚDO DO SITE DEVMEDIA

SEJA UM ASSINANTE MVP!

Tenha acesso garantido a qualquer tema que você possa necessitar!

VIDEOS -AULAS

O assinante MVP acessa todas as vídeo-aulas do site DevMedia. Nenhum outro produto dá essa possibilidade. São mais de **7.000 video-aulas.**

CURSOS

Nosso site possui atualmente **+ de 250 cursos online** com a qualidade que você já conhece e confia. Com o MVP você terá acesso a todos eles.

REVISTAS

Assinante MVP acessa todas as nossas revistas na versão digital, incluindo todas as edições anteriores. Quando pintar aquela dúvida você terá mais de **3.500 artigos para consultar!**

ASSINATURA MVP MOST VALUABLE PROFESSIONAL

A assinatura MVP te dá a oportunidade de aumentar seu conhecimento em diversos assuntos, com qualidade e economia.

Invista na sua carreira se torne um profissional ainda mais completo!

POR APENAS R$59,90 POR MÊS

DevMedia

Acesse: www.devmedia.com.br/mvp